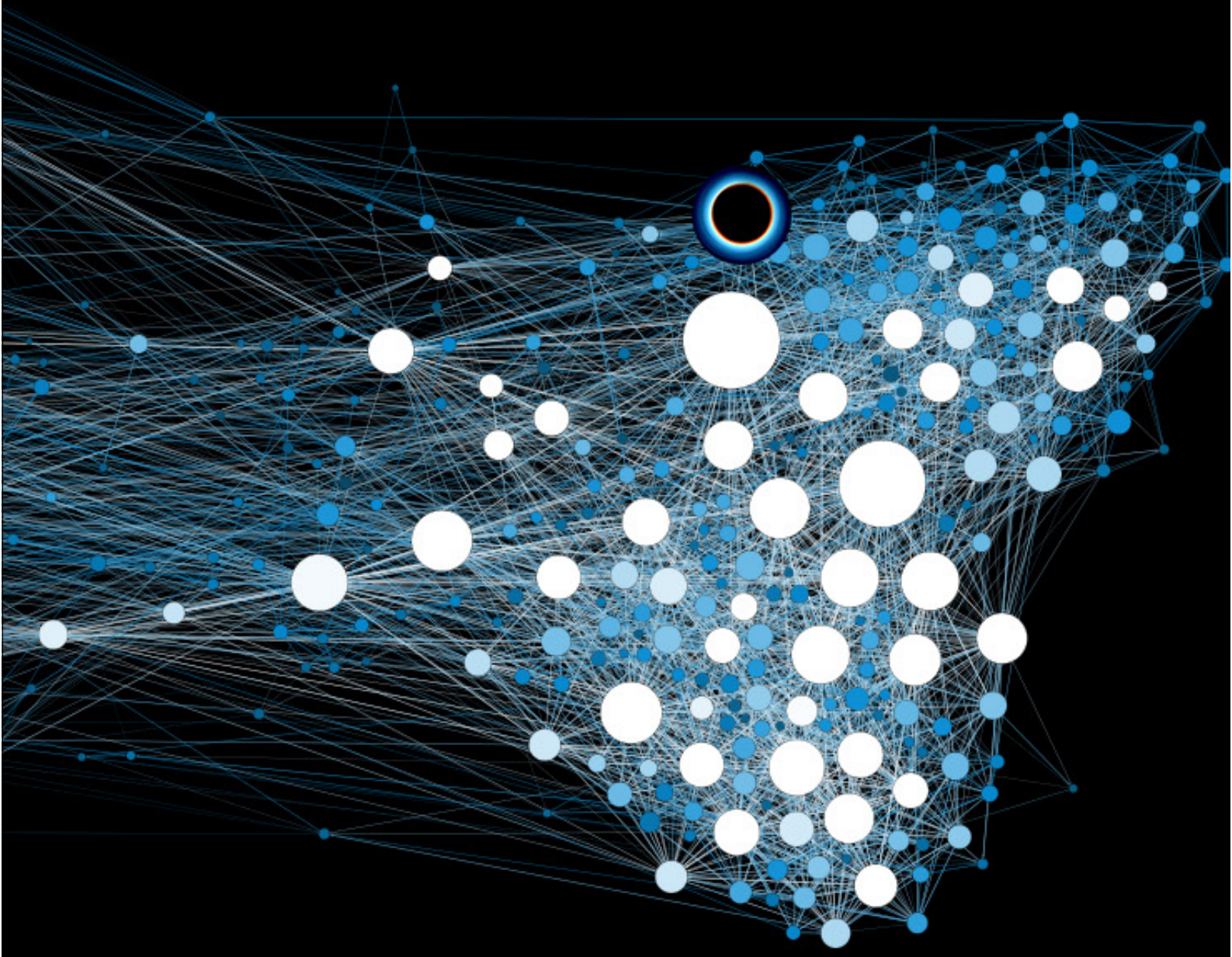
Virtual Internets



Marc X. Makkes

VIRTUAL INTERNETS

2018

VIRTUAL INTERNETS

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het
College voor Promoties ingestelde
commissie, in het openbaar te verdedigen
in de Agnietenkapel
op 15 februari 2018, te 12:00 uur
door

Marc Xander Makkes

geboren te Amsterdam.

Promotor:	prof. dr. R.J. Meijer	Universiteit van Amsterdam
Promotor:	prof. dr. ir. C.T.A.M. de Laat	Universiteit van Amsterdam
Overige Leden:	prof. dr. H. Afsarmanesh	Universiteit van Amsterdam
	prof. dr. ir. H.E. Bal	Vrije Universiteit
	prof. dr. M.T. Bubak	Universiteit van Amsterdam
	prof. dr. S. Klous	Universiteit van Amsterdam
	prof. dr. H. J. van den Herik	Universiteit Leiden
	prof. dr. T.M. van Engers	Universiteit van Amsterdam
	dr. P. Grosso	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Universiteit van Amsterdam
Science Park 904
1098 XH Amsterdam

Cover design by Marc X. Makkes
Cover Image by Martin Grandjean under Creative Commons Attribution-Share Alike 4.0 International license, modified by Marc X. Makkes.
ISBN: 978-94-028-0925-1
NUR code: 988

Dedicated to Ilse, Julia and Luyt

CONTENTS

1 INTRODUCTION

The Internet connects countless networks of computers, anywhere on Earth. Computers come in the form of smartphones, tablets, servers and as PC's. The Internet also connects the computerized parts of cars, machines, homes, and increasingly, the computerized parts of ourselves. The situation fuels our imagination resulting in visions of the Internet of Things (IoT). Figure 1 shows a popular presentation of the IoT. Nearly everything and everybody is interconnected via the internet to computer programs running in data centers commonly referred to as "the cloud". Presently, the abstract vision of the Internet of Things is clearly established as well as its abstract benefits and threats. Figure 1 shows an intuitive grouping in application domains. Virtual internets, the subject of this thesis provides the underlying structures that enables only the desired interactions. A virtual internet is an emulated computer network, that contains internet services, in the cloud. More precisely, a virtual internet is a construction of IP-tunnels overlaying the Internet and networking software, e.g., routers, running on emulated computer hardware called virtual machines (VMs).

The interaction of computers with things in our physical environment is not straightforward. The interaction has to avoid physical damage, violation of laws and respect privacy. And it has to deal with lots of data. At the moment there is a focus of scientists and engineers to extract useful information from enormous amounts of data. This research, indicated with the phrase 'Big Data', is flourishing. The Big Data research doubts that we, humans, are capable of finding the valuable information in the Big Data. Therefore much of the Big Data research attempts to deliver artificial intelligence (AI) capabilities, e.g., to optimize the car traffic on a highway. This sounds fascinating, yet for the IoT even basic ICT capabilities have to be developed. For instance, those to keep data secure at a certain place and those to prohibit the copying of data whilst making it available to software that has the right to process it.

IoT systems can be quite complicated. Take traffic control, where self-driving cars are instructed by a software running in cloud datacenters to take certain actions. There are many parties interacting with this software: consumers, businesses, car manufacturers, telecommunication companies, governments, banks, Internet companies, etc. Data must be shared with some companies and should not be shared with others. There might be a law that prohibits that certain car-traffic data leaves the country. Furthermore, the law may rule that only cer-

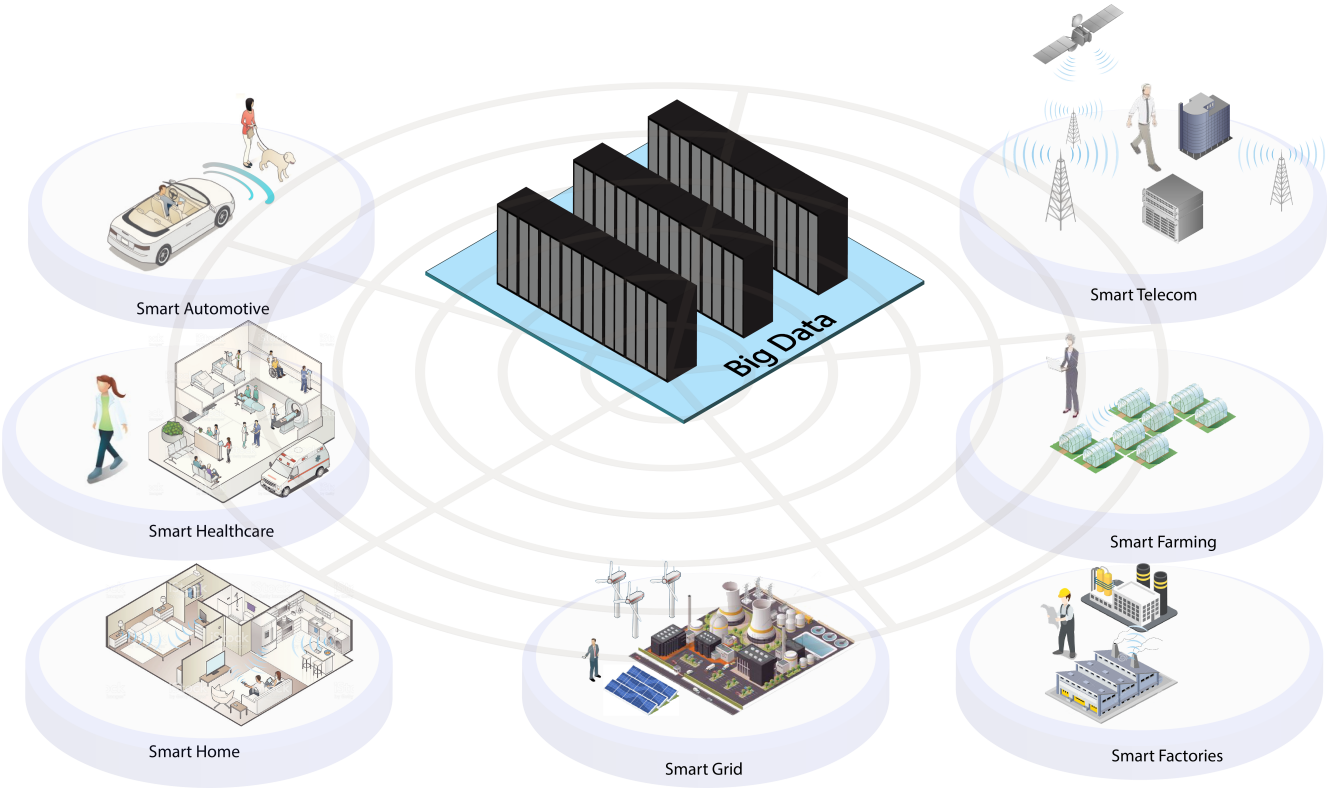


Figure 1: A simplified depiction of the Internet of Things and because of that, it clearly shows the current popular perception about the subject. Applications, e.g., the smart home, are ordered congruently to the domains in the physical world. Connectivity between the "things" has become a technicality, whilst secure dealing with data becomes a key issue. Consequently, data is drawn prominently at the center of the figure as computer racks in a cloud data center. This thesis contributes to enable secure data manipulations and secure data sharing on basis of virtual internets running inside cloud data centers.

tified software is allowed to operate on the data. Some information must be deleted, for privacy sake, after a few weeks. And in case of an accident, data will be exchanged between cars, government institutions, insurance companies, etc. What about cyber attacks? Can hackers stop a street full of cars? The complexity of the self-driving car case hints that, ultimately, software of thousands of companies, organizations and government are involved. An important question is how to organize that, i.e., how to organize the Internet of Things?

1.1 VIRTUAL INTERNETS

This thesis discusses a set of key concepts that have been developed to support the case of virtual internets being the main building blocks to construct IoT applications. As stated before and detailed to great extent later, a virtual internet contains computer emulated, hence virtual, network equipment and computer hardware. Nevertheless, these emulations are able to run regular network and computer software. As part of R&D of software defined networks, our research group at the University of Amsterdam noticed already in 2005[1] that one could create virtual internets. Virtual internets would, just as its real counterpart, be able to transport information between computer programs, e.g., webservers and webbrowsers. In fact, most software that uses internet will work with a virtual internet too. There is, however, a crucial difference between the real and a virtual internet. A virtual internet can be placed under full software control. We call such controlling programs *Netapps*.

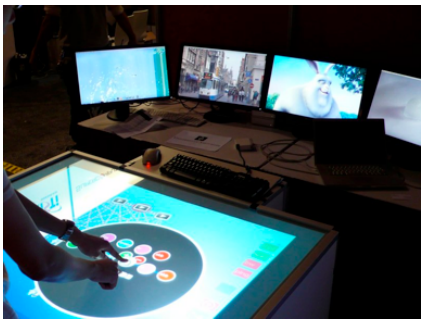


Figure 2: Interactive Networks demonstration at the SuperComputing Conference 2010. This was one of the first proof of concepts that demonstrated the usefulness of software manipulations of virtual internets and the ease of implementing them.

Software, e.g., the video streaming program active in Figure 2 interworks with a virtual internet in the same way as with the real Internet. Yet, the virtual internets are emulated subnets, and therefore software constructs. Hence, they can be manipulated by other software. Indeed, the internet video streaming program, depicted in Figure 2 did not notice the software manipulations active on the virtual internet it was connected to. These manipulations can change however the results of the application. In Figure 2 the manipulations pinched the

transmission capacity on some of the virtual internet links and multicasted the video stream. Furthermore, software manipulations on virtual internets allow features not possible in the normal internet. For example: the manipulations could even force the video packets to loop several times on the same physical connections and guides them to reach the video players. In the real internet, looping packets are dropped. We distributed virtual internets globally by using data centers that are spread around the world. Since virtual internets can be software manipulated they are much more versatile than the real one. At the Supercomputing [**P-5, P-8**] 2011 and 2012 fairs we showed how Netapps dynamically repaired broken virtual internet links. Because most applications are able to deal with temporary internet hiccups the repair of the network did not require a reconfiguration of the connected applications.

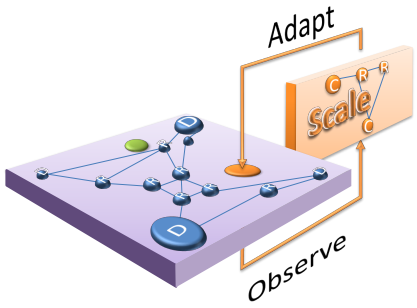


Figure 3: A virtual internet is drawn as a sandbox, implying that it secures the ICT processes it contains by isolating them from the real Internet. Furthermore, a Netapp that manipulates the virtual internet is drawn externally, indicating a separation of concerns. Scaling and distribution, a common concern, is implemented by the Netapp whilst the specifics of the application is implemented by domain experts. As a common concern, scaling and distribution by Netapps is one of the topics of this study.

Our demonstrations illustrated the potential of software control of virtual internets to quite an extent. From these we learned [2] to discriminate generic and specific functions to create virtual internets. Software, named 'Sarastro', was described that implements these generic functions. Furthermore, the Netapps implement the application specific functions, to control, via Sarastro, the lifecycle, scaling and distribution of applications, VMs and virtual internets. We also reported in [2] about a world spanning virtual internet being a network with 163 routers in a line topology. By simply running the Netapp again, a second one could be generated. One might say that Netapps create SDI – software defined internet, paralleling SDN – Software Defined Networks. The SDI consists of emulated computer networks that interwork via the Internet Protocol. Virtual internets are represented in visualizations by the graph shown in Figure 3.

A later insight yielded additional benefits of software control of virtual internets. Inspired by the ability of Netapps to create self-

repairing, globe spanning virtual internets, we applied for the funding of the SARNET project[1], that now investigates the adaptive response to cyber threats on a virtual internet. In the creation of the SARNET proposal, we realized that we can control the exposure of virtual internets to unwanted interactions. The basic mechanism for this is to control the topology and location of the virtual internet with the effect to keep its existence secret and prevent deliberate attacks. Similarly, if an attack is detected one can move parts or the whole virtual internet to other locations to obfuscate its whereabouts.

Virtual internets can be setup to facilitate the controlled interworking of distributed applications originating from multiple companies, for example to facilitate a controlled way to share data. Furthermore, virtual internets allow to setup connections per application to the end-systems of users, e.g., isolated connections to online banking applications. Such sandboxed (see Figure 3) applications can be created on a national or even on an international scale, by using a globally distributed set of datacenters. Hence, virtual internets and (sandboxed) applications that use them can gain a relevant scale and distribution for the IoT.

Such insights we did not have in 2009, the period in which the initial subjects for this thesis research where formulated. In 2009, a virtual internet prototype[3] existed that exposed an interface to manipulate IP-headers by means of a multi-touch table. This prototype facilitated interactive routing in the virtual internet. The prototype inspired the thought that replacement of multi-touch software by application software (that we presently call Netapps) would allow automatized adaption of virtual internets. It was foreseen that such Netapps would automatically scale, distribute and reconfigure virtual internet topologies.

1.2 RESEARCH QUESTIONS

In 2011, at the beginning of this research, the benefits of deploying virtual internets became clear. Yet all benefits of the software control of virtual internets become insignificant if the virtual internet does not perform or costs too much. Cost, in financial and ecological terms, is crucial as virtual internets can become very large. Hence, the motivation of our research is to establish the foundations of security, performance, distribution, cost, and scaling for virtual internets.

The aim of this thesis is to develop the computer science of the constructions needed to make virtual internets a secure and practical environment to assemble and operate a distributed application. In essence we answer the following overall research question:

1 https://sarnet.uvalight.net/

Can we scale, distribute and adapt virtual internets and embed applications in them to achieve a better than best-effort performance of the distributed application?

To answer this question, we have to address others. Applications in virtual Internets experience a location transparency. Yet, in practice, such transparency is not in all cases a desired property. A Netapp instantiating a global scale virtual internet may not simply copy personal data of Dutch citizens to Tokyo. For persons, companies, businesses and the law it matters where data is, what that data is, which software operates on it, who has instructed that operation, if there is proof of execution, where the operation has taken place, where the results are sent to and so on. Hence, the first research question is:

1. How can we entangle virtual and physical machines, and how can we use this entanglement for secure communication purposes?

This question is answered in Chapter 2 by extending the concept of physical unclonable functions (PUFs). PUFs are random hash functions, which are constructed from unique physical parts of the system. The PUFs are unique for a computer and are usually embedded in a CPU. Using the cPUF concepts described of Chapter 2 we can nail down the exact CPU on which software should run, certify that this is the intended software and can generate proof of their execution. In case data is processed, the results can be securely transported via the virtual internet links (VPNs), which are setup using unique device information.

Another important property of virtual internets is performance. The usability of virtual internets gets a plus if their global data transport qualities are comparable or better than that of Internet itself. Examples of such qualities are latency/round trip times, capacity, jitter, etc. As virtual internets are emulated computer networks and since emulations add extra processing steps to data transport processes, loss of performance is to be expected. On the other hand, performance gain can be expected as Netapps allow to create optimized global virtual internet topologies, e.g., by continuously adapting the set of data centers that hosts the virtual internet, or by creating multiple transmission paths. Chapter 3 addresses the issue of topological optimization of virtual internets, answering the following question:

2. Does the optimization of a virtual internet topology result in a better end-to-end performance compared to the best effort path over the Internet?

The software routers deployed in virtual internets run in VMs. VM technology is quite mature and the modest (about 2-5%) performance

penalties are for many applications outweighed by the benefits virtualization brings. In case that future internet traffic runs predominantly via virtual routers in data centers, packet delay matters very much. In high-end routers the lookup is performed by dedicated hardware. Such hardware is not present on computers that run VMs. In present day computers programs are stalled by accessing memory outside CPU. This stalling effect is called the memory wall. The main performance bottleneck of software routers is the lookup of the routing table items in memory. In virtual internets one wants to combine advantages of VMs and the customization ability of software routers with the packet-forwarding speed of dedicated hardware. In Chapter 4 the idea is researched to make routing-table lookup computational bound, by transforming routing table data to CPU instructions. This makes routing almost entirely a CPU activity. Hence, Chapter 4 addresses:

3. Can we improve the process of routing table lookup such that it does not suffer from the memory wall?

For datacenters the availability of 'green energy' is an important factor. Hence, also the ecological footprint of computation matters. We regard this as another cost aspect of computing. As IoT applications can deploy large virtual internets, their ecological footprint must be minimized constantly. In Chapter 5 it is investigated how the total CO_2 emitted by the virtual internet (networking elements and executing applications) can be estimated and be used as input to Netapps.

4. How can we quantify the CO_2 footprint of a virtual internet?

With the methods presented in Chapter 5 one can determine for a given CO_2 cost the scaling and distribution options of the virtual internet. Scaling a distributed application is investigated in Chapter 6. That chapter reports about a case study where a Netapp optimizes the performance of computing resources of a distributed application. The Netapp keeps the distributed application free of bottlenecks, even if the data that it processes features varying qualities. Chapter 6 addresses the following research question:

5. How to scale, for a fixed set of VMs, distributed applications to achieve an optimum performance?

Chapter 6 is relevant for developing generic scaling and distribution services for distributed applications, because this chapter address common concerns: network performance, environmental impact, and application performance.

1.3 THE IV-E PROJECT

The research presented in this dissertation has been carried out in the subproject *programmable infrastructures* (PIF), of the COMMIT IV-e project (e-Infrastructure Virtualization for e-Science Applications)[2]. The aim of PIF is to give e-Science, i.e., the computationally intensive science that is carried out in highly distributed network environments, the capability to continuously adapt the service of the network and computing resources for an optimal performance of specific virtual laboratory experiments. PIF provided the issues that were tackled in this thesis. Most noticeably is the implementation of the "Golden Demo" [**P-**3]. Its goal was to demonstrate integration of results of the COMMIT IV-e subprojects to scientific and non-scientific attendances. The virtual internet technology developed for this thesis enabled an easy going application integration. Scaling, distribution and robustness, was then implemented via Netapps as discussed in Chapter 6.

The remainder of this dissertation is structured as follows: In Chapter 2, a new concept for PUFs and their usage is presented. In Chapter 3 a router distribution framework, named Metro, is developed and studied. A novel, CPU bound routing-table lookup algorithm is investigated in 4. Chapter 5 introduces a CO_2 cost model that can be used in a decision framework to determine an optimal distribution of the virtual internet. Chapter 6 deals with automatic scaling and distribution of application parts within virtual internets. Finally, Chapter 7 revisits the research questions, summarizes the results and concludes. Chapter 7 also presents advanced applications that this thesis enables and defends the theorem that, in future, Internet will be predominantly a distributed application in the clouds.

2 http://www.commit-nl.nl/projects/e-infrastructure-virtualization-for-e-science-applications

2

ESTABLISHING SECURITY ON UNIQUE PHYSICAL CHARACTERISTICS OF CPUS

This chapter is based on "Flowchart description of security primitives for controlled physical unclonable functions" [4].

Most people have their email server in a cloud with an online copy of their emails. It matters if somebody moves it to another country. Although it might have a better performance at its new location, it might be less protected to prying eyes. If the email server is part of a virtual internet, the subject of this thesis, then a Netapp might decide, or not, to move it. To see if an email server can be moved to another processor, a Netapp has to identify the processor, and would use a table to determine if the processor is in the right country. Clearly, the processors ID must be unique and non-hackable. Besides the identification of computer systems, some IoT applications also require identification methods for sensors and actuators. This chapter describes cryptographic methods, implemented as service primitives, with which Netapps can securely verify a processors identity. On basis of that, other service primitives are presented to set up secure communication channels and to obtain proof of execution of software that the processor runs.

To identify a modern integrated circuit, one uses physical unclonable functions (PUF). PUFs are based on the intrinsic properties of a part of the circuit, e.g., a unique and complex mesh of resistive wires. On-chip circuits measure the resistance values and report them to the processor. Packed securely in the chip, it is practically impossible to measure the resistances without destroying the chip. Furthermore, the chip has to include electronics that allows software to interact with the PUFs.

We enhanced the PUF concept to make it a safe mechanism for distributed computing environments that are susceptible to model attacks. Model attacks deploy techniques to replicate the behaviour of a PUF by letting artificial intelligence algorithms learn corresponding input (challenge) and output (response) pairs. The paper presents the design of a control layer that protects PUFs against model attacks and allows secure (Internet) communications to interact with the PUF. The concept is called c(ontrolled)PUF. In addition, we model five security primitives, which are: bootstrapping, secure channel setup, renewal, proof-of-execution and certified execution.

In Chapter 7 we describe a distributed, secure digital market place for transactions on data. In essence, digital market places are an execution environment in which certified programs operate on and exchange data. One of the most secure ways an execution environment can identify intended hardware and provide proof that output is from the intended software, is to use cPUFs. A way to do this is to deliver Netapps a listing of cPUFs and their locations and other properties. On basis of such lists NetApps can setup the

distributed system by instantiating certified software and their certified execution environment, acting also as a sandbox, on a computer with the right cPUF. This then would allow other Netapps to verify that output stems from certified software that is executed on a specific cPUF.

2.1 INTRODUCTION

2.1.1 *Physical unclonable functions*

The concept of physical unclonable functions (PUFs), also known as physical one-way functions or physical random functions, was introduced in [5] . A PUF was originally defined as a physical object with the following properties: (1) It can be challenged by applying a stimulus to it, and the responses are highly unpredictable and unique to each object. Applying a challenge and measuring the response can be done efficiently. The number of challenge-response pairs is very large. (2) The object is hard to clone physically, even by the original manufacturer. (3) It is hard to model mathematically.

Some physical systems are referred to as PUFs even though they do not satisfy all these properties. Controlled PUFs can be realized from physical structures with less stringent properties.

A good example of PUFs are the Optical PUFs introduced in [5] . These consist of a transparent material containing scattering particles at random locations. When laser light is shone onto it, coherent multiple scattering occurs. An image made of the reflected or transmitted light shows a so-called speckle pattern, a highly irregular pattern of bright and dark spots. The pattern is highly sensitive both to the locations of the scattering particles and to the properties of the incoming laser light, such as wavelength, angle of incidence and focal distance. The angle of incidence, for instance, can be used as a 'challenge' to the PUF. The resulting speckle pattern has a large entropy [6, 7] and can be seen as the 'response' to the challenge.

2.1.2 *Authentication using bare PUFs*

Originally the use of PUFs was envisaged for authentication in the following manner. An Optical PUF supports a very large number of such challenge-response pairs (CRPs). Furthermore, knowledge of a large set of CRPs gives only negligible information about the response to a new challenge [8]. In [5] it was proposed to use PUFs as remote authentication tokens. PUFs are randomly manufactured by the verifier, Alice. The following procedure is followed independently for each PUF.

- In the enrolment phase, Alice generates a number of random challenges. She measures the response for each challenge and

stores the set of CRPs for that PUF in a database. The PUF is then handed over to a user, Bob. Alice couples users to PUF identifiers in her database.

- In the verification phase, Bob wishes to prove to Alice that he possesses a specific PUF. He sends the PUF identifier to Alice. Alice looks up the CRP list for this specific PUF in the database. From the list she randomly selects a CRP. She sends the challenge part of the CRP to Bob. He applies the challenge to the PUF and measures the response. He sends the response to Alice. She compares Bob's response to the response in her database. If these match, then Alice is convinced of the PUF's authenticity. Whatever the outcome, the used CRP is removed from the list.

Alternatively, Bob does not send the response in the clear to Alice. Instead, the response is used to derive a shared secret between Alice and Bob, which they then use for an authentication protocol 1. Optionally, a session key is generated from the shared secret as well. The security of the PUF as an authentication token as described above completely depends on the unclonability of the PUF and the unpredictability of its responses.

2.1.3 *Dealing with measurement noise*

Apart from Optical PUFs many other types of PUF technology have been described in the literature, such as reflection of laser light from paper fibers [9], randomized dielectrics in protective chip coatings [10], radiofrequent responses from pieces of metal [11] or thin-film resonators [12], delay times in chip components [13] and start-up values of SRAM cells [14]. In this chapter we will not be concerned with the physical aspects of PUFs, but merely assume that PUFs are available as a resource with all the right properties.

Whatever the physical realization of the PUF concept, there is a common problem that needs to be solved: noise in the response. The measurements are analog and hence inevitably noisy. A measurement result cannot be directly used in a cryptographic primitive such as a one-way hash or a block cipher. A single bit flip in the input (due to noise) would result in roughly 50% bit flips in the output. Hence, an error-correction step is needed so that Alice and Bob can exactly agree on the same bit string representation of a PUF response. (This is known as information reconciliation). However, the error-correction is nontrivial. The usual attacker model for PUFs assumes that the redundancy data which is required for noise elimination is known to the attacker. Hence it is necessary to make sure that the redundancy data does not leak critical information about the common secret (the "key") derived from the response. The concept of a Fuzzy Extractor [15, 16], also known as a helper data scheme [17], was introduced

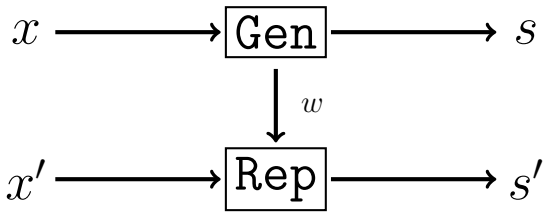


Figure 4: Fuzzy Extractor. The Gen function takes a measurement X as input, and generates helper data w and a near-uniform key S. The Rep function attempts to reproduce S from w and a noisy measurement X′. It succeeds ($S' = S$) if the noise is sufficiently weak.

as a primitive that achieves both information reconciliation and privacy amplification. The redundancy data (called helper data in such schemes) suffices to reproducibly reconstruct a string from noisy measurements (see Fig. 4), yet leaks only a negligible amount of information about the extracted key. In this chapter we will not be concerned with the exact details of fuzzy extractors. We will merely assume that proper helper data is present.

2.1.4 *Controlled PUFs*

The concept of a Controlled PUF (cPUF) was introduced in [18]. A cPUF is a combination of a PUF and a control layer in which the PUF is inseparably embedded. The control layer completely shields off the PUF inputs and outputs from the outside world. Any communication with the PUF has to occur through the control layer electronics. Any attempt to force the components apart will damage the PUF. A cPUF has considerably stronger security than a bare, unprotected PUF, since attackers cannot probe and query the PUF at will. In effect, the cPUF is a sort of trusted computing environment. The term "Physically Obfuscated Key" (POK) was coined by Gassend for this type of stored key. In contrast to the scenario sketched in Section 2.1.2, a POK key does not need to be discarded after use. A POK does not need all the PUF properties listed in Section 2.1.1 Instead, it only requires that the key obtained by the fuzzy extractor is shielded off from attackers. For a cPUF it is furthermore required that there are sufficiently many CRPs to accommodate all users, but by no means the "very large" number mentioned in Property 1.

In Gassend et al. [18, 19, 20] a way was presented to employ this trusted environment for the purpose of outsourcing computations. The idea is roughly as follows. (More details are given in Section 2.3.)

First, CRPs of the PUF are handed to users in a secure way. Everybody (even people without CRPs) can remotely run programs on the cPUF control layer. There is a special Application Programming Interface (API) for accessing the PUF. With the help of this API a user can instruct the cPUF to generate a 'proof' of the correct execution of the outsourced program. This proof can be thought of as a message authentication code (MAC) over the executed program and the program output, using the PUF response as the MAC key. If the user has a valid CRP, he can verify the MAC. (See Section 2.3.2.3) This procedure is referred to as 'certified execution'. In the construction of [18] the proof is verifiable only by the user who sends the task to the cPUF. In [20] this was generalized to a proof ('E-proof') that can be verified by third parties as well.

The above scheme provides a way for users to outsource computations and be certain that their program was correctly executed, by the designated device, yielding the given result. No public key infrastructure is needed. Instead, the security is based on the secrecy of the CRPs. In addition to the proof generation, [18, 19, 20] also provide a number of protocols for CRP management, most notably bootstrapping (creation of the original CRPs) and renewal (allowing a user who possesses a CRP to obtain more CRPs). For an overview of PUFs, cPUFs, and fuzzy extractors we refer to [19].

2.1.5 *Contributions*

In this chapter we propose a modification of the main cPUF security primitives. We work with the same assumptions about the PUF as in the original literature on cPUFs, and the same way of challenging the physical structure. Our modifications of the primitives improve the overall security by putting additional restrictions on access to the cPUF and by encrypting more of the exchanged messages. We represent the protocols in a different way from [18, 19, 20], namely in the form of flowcharts, which improves the comprehensibility of the protocols and of their security properties.

In [18, 19, 20] the protocols between users and a cPUF were represented as programs executed by the cPUF's control layer, using a specific security API. Hashes of these programs play an important role in the security primitives. In some cases, a function call involves a hash of a piece of the program containing the function call. We feel that such a formulation is needlessly complicated. Especially the self-referential nature of the program hashes can be confusing. In our flowchart notation, each security primitive corresponds to a 'mode' of the cPUF, in which the control layer has a certain fixed input/output behaviour. A user can instruct a cPUF in which mode to operate, but cannot change the cPUF's sequence of actions in that mode. For each mode we present a flowchart. There are no hashes of con-

trol layer programs; the security clearly derives from the secrecy of he challenge-response pairs. Avoiding the program hashes allows for more efficient implementation.

In contrast to Gassend et al., we do not allow just anybody to outsource computations to the cPUF, but we first demand that a user establishes a secure channel with the control layer, based on a shared CRP. Any further communication has to take place through this channel. The advantage of this approach is twofold: (i) it provides more data confidentiality, e.g., the outsourced job and the results are not revealed to eavesdroppers, and (ii) it restricts the opportunities for attacks.

Finally, we explicitly show how the helper data is handled; this makes no essential difference with respect to the prior literature but completes the data flow overview. The outline of this chapter is as follows. We first explain the attacker model in Section 2.2 and summarize the construction of Gassend et al. in Section 2.3. Then we present our flowchart formulation in Section 2.4. We summarize our results in Section 2.5.

2.2 ATTACKER MODEL

The manufacture and enrolment of cPUFs occurs in a secure environment. In particular, the bootstrapping (Sec. 2.3.2.1 and 2.4.2 is done by a trusted third party (TTP); this procedure yields CRPs. After obtaining a number of CRPs, the TTP disables bootstrapping, i.e., it becomes impossible for any attacker to run the bootstrapping process again, even if the attacker is in physical possession of the cPUF. We do not specify how the disabling is done.

The TTP hands over CRPs to users in a secure way, i.e., there is no eavesdropping. The cPUF is given to Bob (also in a secure way). The assumption is made that programs running on the control layer execute in a private and authentic way, i.e., their internal data is inaccessible to an attacker, and nobody can cause the program to execute incorrectly. Bob runs a remote computation service, offering trusted computing (based on the cPUF) to users. Bob allows users to communicate with his cPUF, but they have no physical access to it. The communication channel is untrusted. Attackers can eavesdrop and manipulate, delete and insert messages. Authentication between a user and the cPUF is based on a shared secret, namely the response to a challenge. Note that some PUF challenges will inevitably be sent in plaintext over the insecure channel. The security is derived from the unpredictability of the responses.

There are three kinds of attackers: (i) users who do not have a valid CRP, (ii) users who have at least one valid CRP, e.g., obtained from the TTP, and (iii) Bob. The following events constitute a successful attack:

- A user, or Bob, learns a response that belongs to another user.
- A user, or Bob, authenticates to the cPUF with a CRP that he does not know.
- An attacker causes a legitimate user to accept a bogus outcome.
- Bob, or a user other than the user who sends the job, learns the outcome of the computation.

2.3 API FORMULATION OF CPUF PRIMITIVES

We first review the main cPUF primitives as described in [18, 19, 20]. In the original papers the security arguments for these primitives were informal; a more thorough proof of security in the attacker model of Section 2.2 was given in [21] using formal methods.

We do not discuss all the protocols, but restrict ourselves to Certified Execution, E-proofs and basic CRP handling (bootstrapping and renewal). We try to cover all the essentials. For missing details we refer to the original literature. We have kept the notation as close as possible to the original. Programs are written in C-like syntax. Variables are declared with 'my' as in Perl.

2.3.1 *Hashblocks*

The control layer maintains a stack containing program hash values. The most recent value pushed onto the stack is also referred to as `PHashReg`. The API has a command `hashblock` which manipulates the stack as follows.

```
hashblock(arg1)(  {
    ... lines of code ...
},arg2)
```

The above code leads to the execution of 'lines of code' and computation of a hash over the concatenation of `arg1` with all the lines of code within the `{}` brackets and `arg2`. When execution reaches the `hashblock` command, the cPUF computes this hash and pushes it onto the stack. When execution reaches the final) brace, the top value of the stack is popped off and purged. Variables declared within a hash block are automatically cleared on exit from the hash block. Note that the code within the hash block has access to `PHashReg`, i.e., a hash over itself. The self-referential nature of this construction was one of our motivations to look for a simplification. The purpose of the hashing and of the rather peculiar structure, with `arg1` and `arg2` sandwiching the program code, is explained in Sec. 2.3.3.

The control layer accesses the PUF through the function '`PUF`' .

- `PUF(Chal)` yields the PUF response to challenge `Chal`.

We list the API commands that deal with PUF access and the PUF responses. These commands are available to users.

- `GetResponse`. This instruction feeds `PHashReg` to the PUF as a challenge. `GetResponse() = PUF(PHashReg)`.
- `GetSecret`. Essentially, this instruction generates a hash of a PUF response. `GetSecret(Chal) = Hash(PHashReg, PUF(Chal))`.

2.3.2 *API notation for CRP handling, certified execution and E-proofs*

2.3.2.1 *Bootstrapping*

The CRP management of a cPUF is bootstrapped in a trusted environment. A trusted third party (TTP), e.g., the manufacturer or a cPUF issuer, obtains the first CRPs from the cPUF by running the following program,

```
Bootstrap(Prechal): hashblock(PreChal)( {
    Return GetResponse();
  });
```

Here 'PreChal' stands for 'pre-challenge'. The above code computes the hash of `PreChal` concatenated with the instruction between `{ }` brackets (the hash that gets stored in `PHashReg`), then feeds that to the PUF and directly returns the PUF output. The TTP has to compute the actual PUF challenge `PHashReg`, and stores it along with the cPUF's output as a CRP.

As bootstrapping gives CRPs to someone who does not yet have a CRP, this function should be disallowed after the TTP has obtained its CRPs.

2.3.2.2 *Renewal*

Users who already have a CRP(`OldChal`, `OldKey`) can obtain more CRPs by running the 'renewal' protocol on the cPUF, as follows.

```
Renew(OldChal,PreChal): hashblock(OldChal,PreChal)( {
  my newR = GetResponse();
  my OldKey = GetSecret(OldChal);
  return EncryptAndMAC(newR,OldKey);
});
```

As in the bootstrapping primitive, `PreChal` is a pre-challenge. It is chosen randomly by the user. The cPUF creates an encrypted channel back to the user through which it sends the new response. The actual new PUF challenge is a program hash that depends on both `PreChal` and `OldChal`. The user computes this hash; together with newR it forms the new CRP.

2.3.2.3 *Certified execution*

The method of creating an encrypted channel back to the user is used also for Certified Execution. A user possesses a CRP (`Chal`,`Resp`). Running the Certified Execution protocol for a job `Prog` with this CRP is done as follows,

```
CertifiedExecution(Chal,Prog): hashblock (Prog)( {
   my result;
   hashblock (){{ result = RunProg(Prog); });
   my key = GetSecret(Chal);
   my cert = (result, MAC(result,key));
   Return cert;
});
```

Here `Runprog(Prog)` stands for execution of Prog on the control layer. The user has all the ingredients to compute key himself, so he can verify the MAC: `key = Hash(PHashReg,Resp)`. The register value PHashReg is known to the user, since he knows Prog, and the above listed code is publicly known.

2.3.2.4 *E-Proof generation*

Next we list the steps for E-proof generation and verification as given in [] .

```
EproofGen(Prog):
my HProg = Hash(Prog); hashblock (HProg)(HCodeA, {
   my result;
   hashblock () ({ result = RunProg(Prog); });
   my secret = GetResponse();
   my Eproof = (result, MAC(result,secret));
   return Eproof;
});
```

The parameter HCodeA stands for the hash over the arbitration program (see Section 2.3.2.5). The PUF challenge for deriving the MAC key is completely determined by `Prog`. Nobody but the cPUF has access to this MAC key.

2.3.2.5 *E-proof verification ('arbitration')*

A verifier who has `Eproof`, `Prog` and a valid CRP can check the correctness of `Eproof`. He first computes `HProg = Hash(Prog)`, and then runs the following arbitration program on the cPUF through Certified Execution.

```
EProofVer(HProg,Eproof): hashblock(HProg)( {
   my (result, M) = Eproof;
   my secret = GetResponse();
   if M = MAC(result,secret)
      return(true);
   else
      return(false);
},HCodeE);
```

Here HCodeE stands for the hash over the E-proof generation program (see Section 2.3.2.4).

2.3.3 *Security purpose of the hash blocks*

We briefly review the purpose of all the `hashblock` instructions, as explained in [18, 19, 20].In the `Bootstrap` procedure it is not strictly necessary to have a `hashblock`, since the TTP sees all secrets, but for simplicity the structure of the program is the same as in Renewal.

In `Renew`, the hashing of the challenge is of crucial importance. If `Renew` were to have a direct challenge `Chal` as its argument, then an attacker could exploit this: The attacker knows a CRP (`Chal1`, `Resp1`). He overhears another user's challenge `Chal2`, e.g., when that user is doing Certified Execution. The attacker would be able to run Renew with `Chal2` as the new challenge (and `Chal1` as the old) to obtain the response `Resp2`. This attack is prevented by the use of a *Pre-challenge* which has to be hashed in order to turn it into a challenge. Knowledge of `Chal2` does not help the attacker to find `PreChal`; he has to compute a hash pre-image. The legitimate user, on the other hand, does know `PreChal`. Hence, the main purpose of the `hashblock` is to turn the pre-challenge into a challenge.

In `CertifiedExecution` and `EProofGen`, the function of the outer `hashblock` is to make sure that `Prog` is not tampered with. The `key` in `CertifiedExecution` and the secret in `EProofGen` are made to depend on `Prog`. The inner `hashblock` ensures that `Prog` does not have access to secrets such as the MAC key. The extra `hashblock` instruction pushes a new value into `PHashReg`, which prevents Prog from accessing the previous `PHashReg` value.

The purpose of the rather peculiar three-part hash construction, with `arg1` and `arg2` sandwiching the program code, allows `iEProofGen` and `EProofVer` to have access to the same secret. This is achieved by letting these two primitives each contain a hash of the program code of the other primitive (`HCodeE` and `HCodeA`); in this way they are 'coupled' together, and no other program can get access to their shared secret.

Furthermore, it is our understanding that in each of the listed programs, the hash blocks are also meant to ensure that the PUF primitives are always run using the exact same lines of code. Users can run

arbitrary code on the control layer, including the PUF commands `PUF`, `GetResponse`, and `GetSecret`, but any malicious modification to the programs in Sections 2.3.2.2-2.3.2.5 leads to modified key values.

2.3.4 *Remarks about the API formulation*

We feel that the API construction is somewhat unsatisfactory from the point of view of implementation efficiency. Furthermore, the security of the protocols is not always transparent. (Note though that no security holes have ever been found).

1. In the work of [18, 20, 22] users are allowed to run any code on the control layer, *including PUF commands*. This is an open invitation for exploits. One of the purposes of the hash blocks is to thwart code modification attacks. We would like to depart from the philosophy that users should be allowed to execute PUF commands at will. Since the number of security primitives involving the PUF is very limited anyway, we suggest to 'freeze' the code that uses them, and allow user access only in the form of calls to `Renew`, `CertifiedExecution`, `EProofGen` and `EProofVer`, which now consist of fixed code in the control layer ROM.

   If this approach is taken, hashing of the control layer code becomes unnecessary. Note that the outsourced jobs `'Prog'` and the pre-challenges must of course still be hashed, but this can be done without hash blocks. Hence, one can get rid of the hash blocks. One advantage of this approach is efficiency: the cPUF no longer has to compute hashes of incoming programs.

2. When a function call to `GetResponse` or `GetSecret` is placed inside a hash block, this leads to the highly selfreferential situation that an instruction operates on a hash over itself. While there is nothing wrong with this per se, it is confusing. The self-reference serves no security purpose other than fixing the code, as mentioned in Section 2.3.3. The confusion is avoided in the approach that we suggest above.

3. The Renewal protocol can be run even by users who do not possess a valid CRP. While this does not immediately pose a security risk (the attacker does not have `OldKey`, so the encrypted `newR` is inaccessible to him), it allows CRP-less attackers to run a sort of denial of service (DoS) attack: they may overwhelm the cPUF with requests for Renewal, even though they are not entitled to Renewal.

4. In the Renewal protocol, the `PreChal` is sent in the clear, and any eavesdropper can compute the actual challenge to the PUF.

While this is not a security risk, the leakage of the new challenge could easily have been avoided by a slight change to the Renewal protocol: the pre-challenge could be sent to the cPUF over a secure channel, based on the shared secret `OldKey`.

5. E-proof generation and verification is 'asymmetric' in the sense that anybody can initiate E-proof generation, but a valid CRP is needed for E-proof verification. Again this opens up the possibility of a DoS attack by CRP-less attackers. They can overwhelm the cPUF with jobs to be run in `EProofGen`.

6. In Certified Execution and E-proof generation, the `Prog` and result are communicated in plaintext. While this does not necessarily have to be considered as a security risk, it would have been easy to build in some extra confidentiality: Again, it would have sufficed to set up a bidirectional secure channel based on a shared secret (the PUF response).

2.4 PROTOCOL MODIFICATIONS AND FLOWCHART REPRESENTATION

2.4.1 *Our improvements*

In this section we introduce a more transparent representation for specifying user interaction with a Controlled PUF. The approach is based on Remark 1 in Section 2.3.4. We disallow arbitrary execution of PUF commands. Since the lines of code in the Bootstrapping, Renewal, Certified Execution and E-proof programs are then fixed anyway, we may as well replace these programs by fixed circuits.

In this way we remove the self-referential nature of the `GetResponse` and `GetSecret` function calls, while at the same time improving efficiency by reducing the amount of hashing.

Each circuit (flowchart) corresponds to a 'mode' of the cPUF. A user can instruct a cPUF in which mode to operate, but cannot change the cPUF's sequence of actions in that mode.

We furthermore completely 'symmetrize' all the interactions between the cPUF and a user. We introduce a basic protocol underlying all the others: the setup of a (bidirectional) secure channel (SC) based on the shared knowledge of a CRP. We demand that any cPUF protocol has to run through a SC, i.e. a user needs a valid CRP in order to achieve any further communication with the cPUF whatsoever. This reduces the potency of DoS attacks and provides more confidentiality of challenges, outsourced jobs and results than the construction of Gassend et al. Hence our protocols do not have any of the drawbacks listed in Sec. 2.3.4.

Our construction immediately leads to a substantial simplification: Execution of any user program by the control layer is automatically

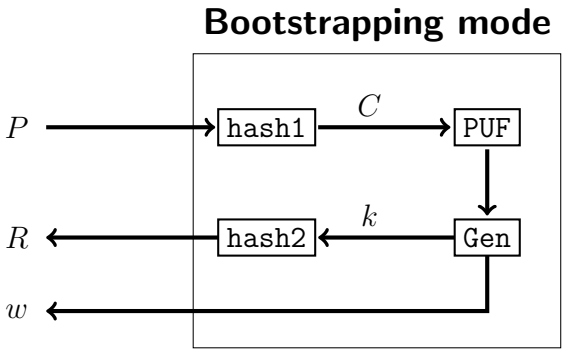


Figure 5: Bootstrapping mode. The control layer receives a pre-challenge P. The pre-challenge is hashed, yielding a challenge C, which is fed to the PUF. The PUF output is sent to the Gen function, which generates a secret key k and helper data w. The key k is hashed, yielding the response R. Finally, the control layer outputs the helper data w and the response R

Certified Execution. Therefore we do not need a separate flowchart for Certified Execution. As a final technicality, we explicitly include the handling of the PUF helper data in our flowcharts. While this does not add anything essential to the protocols, it completes the visualization of all the data flows and clearly indicates which PUF processing (Gen/Rep) occurs where.

In Sections 2.4.2-2.4.5 we present our flowcharts for Bootstrapping, SC setup, CRP Renewal and E-proof generation and verification. The shaded area in each flowchart represents actions that occur within the control layer. A block arrow indicates data sent through a secure channel.

2.4.2 *Flowchart for bootstrapping (Fig. 5)*

The CRP management of a cPUF is bootstrapped in a trusted environment. A trusted party, e.g., the manufacturer or a cPUF issuer, obtains the first CRPs from the cPUF in bootstrapping mode[1] . These CRPs4[2] {C, w, R} are distributed to authorized users. Bootstrapping is the only time at which the control layer ever reveals a PUF response in the clear to the outside world. After the trusted party has obtained a number of CRPs he permanently disables the bootstrapping mode.

1 The 'hash1' function is included here for cosmetic reasons only, in order to have exactly the same flowchart as for our Renewal protocol. Its role will become apparent in Section 2.4.4. Note that hash1 and hash2 are different hash functions. The output of hash1 is a PUF challenge, while the output of hash2 is a key. The role of 'hash2' is to ensure that there is a secret known only to the control layer. This is important for the E-proofs (see Sections 2.4.4 and 2.4.5) .

2 The PUF challenge C and the helper data w together are considered as a challenge to the cPUF.

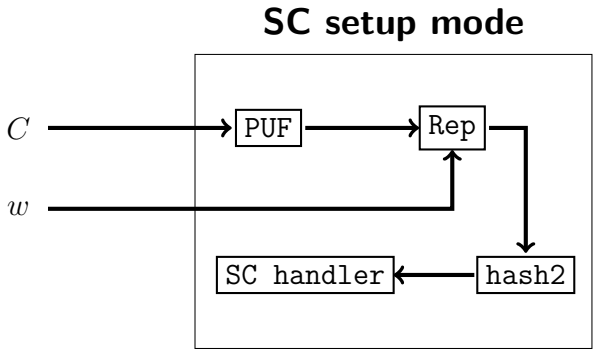


Figure 6: Secure Channel setup mode. The control layer receives C and w. It feeds C to the PUF. The PUF output and w are sent to the Rep function, which reproduces the key k. This gets hashed, yielding the shared secret R, which is then used by the 'SC handler' module to handle the secure communication channel with the user.

2.4.3 *Flowchart for secure channel setup (Fig. 6)*

user who possesses a CRP for a specific cPUF can setup a secure channel with that cPUF over an insecure communication line. See Fig. 6. The security is based on the fact that the response R is secret, even though C and w are revealed to attackers. The shared secret R allows the user and the cPUF to encrypt their communication, generate MACs etc. In Fig. 6 we have deliberately abstracted away the details of the SC setup by putting everything in a box called 'SC handling'. Many ways are known to establish a SC and then to properly communicate through it (with protection against replay attacks etc.), so we do not have to be specific here.

2.4.3.1 *Flowchart for CRP renewal (Fig. 7)*

Any user who already possesses a valid CRP for a certain cPUF can obtain additional CRPs for that cPUF using Renewal mode. Our flowchart for Renewal (Fig. 7) is very simple: it amounts to Bootstrapping executed through a Secure Channel. The user first establishes a SC with the cPUF. Then he initiates renewal mode. He sends a random pre-challenge Pnew and receives Rnew, wnew. Finally he computes C_{new} =hash1(P_{new}) and stores $\{C_{new}, w_{new}, R_{new}\}$.

Remark Similar to the Gassend et al. construction, man-in-the-middle attacks are prevented by the fact that the hash1 function is present at Renewal, but not at SC setup. This prevents an attacker from abusing Renewal to obtain the response R for eavesdropped challenges (C,w). He would have to invert hash1 to obtain the proper pre-challenge $P = \text{hash1}^{inv}(C)$.

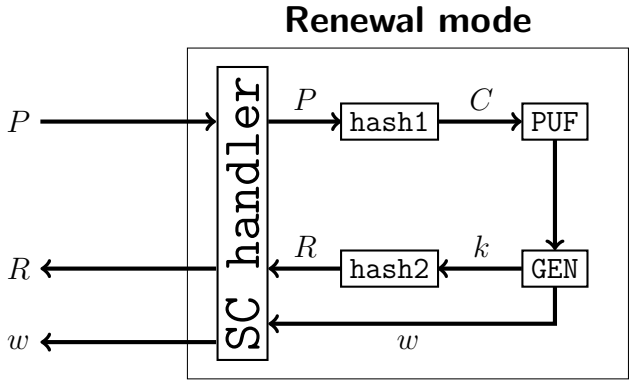


Figure 7: Renewal mode. The steps in the cPUF are identical to Boot- strapping, but communication between the user and the cPUF goes through a secure channel

2.4.4 *Flowchart for E-Proof generation (Fig. 8)*

We present our variant of E-proofs (verifiable by third parties). The protocol is run through a SC based on a CRP $\{C, w, R\}$. A user Alice outsources the execution of a program prog to the cPUF. She receives the result of the computation and the proof Eproof. She stores $\{C, w,$`prog,res,Eproof`$\}$ for later use. The MakeProof module in Fig. 8 can be, e.g., a MAC using k as the key, or a keyed hash. The security is based on the fact that the 'internal' secret key k is known only to the cPUF. Hence nobody is able to forge the certificate, not even Alice, who has $R =$ `hash2`(k), or even the trusted enrolment authority.

Remark. The only program hash occurring in the E-proof generation is the hash over the to-be-executed job. There are no hashes over API instructions as in Gassend et al.

2.4.5 *Flowchart for E-proof verification (Fig. 9)*

When user Alice wants to convince a third party, Victor, that prog executed on the cPUF gave the result res, she hands over to Victor the data $\{C, w,$ `prog, res, Eproof`$\}$. Victor establishes a SC with the cPUF using one of his own CRPs. Through this SC he runs the E-proof verification protocol. The protocol amounts to nothing more than checking the consistency between the E-proof, the 'certified' data $\{C, w,$`hash3(prog), res`$\}$ and the key k. If the E-proof is a MAC as in the example above, then the consistency check is a simple MAC verification.

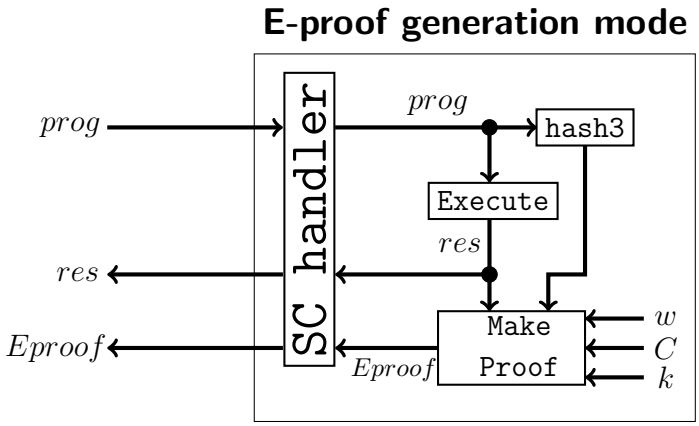


Figure 8: E-proof generation mode. The protocol is run through a SC. The SC-key in use is the hash of the secret key k; this k never leaves the cPUF. The user sends a program, which is executed and also hashed by the cPUF. The key k is used by the MakeProof function to certify the program hash, the result of the computation, and the SC setup parameters C, w

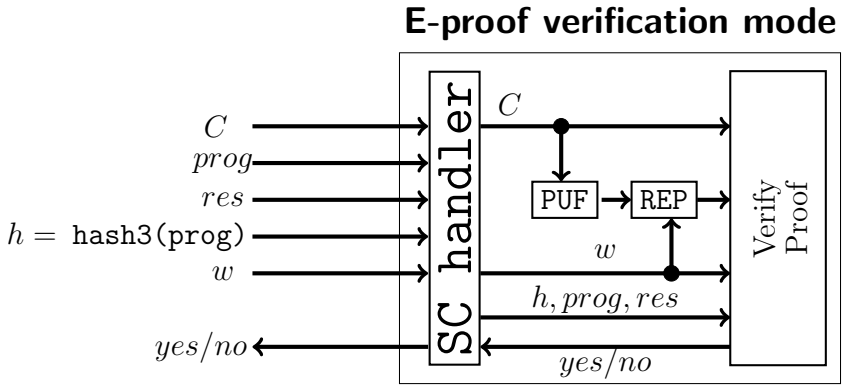


Figure 9: E-proof verification mode. The protocol is run through a SC. The cPUF recovers the secret key k from (C, w) by using the PUF and the Rep function. The `VerifyProof` module then verifies the consistency between `Eproof`, the key k and the data C, w, h, `res`

2.5 CONCLUSION

We have given a modified version of the basic cPUF protocols for CRP management, certified execution and proof of execution. Our modifications further restrict access to the cPUF, and provide more confidentiality. We have introduced flowchart notation to replace the API program formulation of Gassend et al. This gets rid of the selfreferential program hashes and clarifies the essential steps of the protocols. The security clearly derives from the secrecy of the challenge-response pairs. Furthermore, elimination of the program hashes reduces the amount of work done by the control layer.

In our flowchart notation, each security primitive corresponds to a 'mode' of the cPUF, in which the control layer has a certain fixed input/output behaviour. A user can instruct a cPUF in which mode to operate, but cannot change the cPUF's sequence of actions in that mode. Finally, we have explicitly shown how the helper data is handled, completing the data flow overview.

3

FINDING THE LOWEST LATENCY GLOBAL END-TO-END ROUTE VIA THE CLOUD

This Chapter is based on "Metro: Low latency network paths with routers-on-demand"[23].

This Chapter shows how Netapps can optimize the packet transport QoS between endpoints linked by virtual internets. It does so by forcing packets to travel via certain data centers that host the networking elements, e.g., routers, of the virtual internets. In our study, the endpoints and data centers are globally distributed. In more than half of the cases the optimizations resulted in better QoS of the virtual internet connections compared to the default internet path between the endpoints. As the optimization goals are applicable for most networking applications, it makes sense to implement the path-optimization algorithm as a generic service that a Netapp uses to optimize a virtual internet. This would simplify the design and development of the distributed application that uses the virtual internet. Beyond that, the work presented in this Chapter states that virtual internets extend the implementation for Internet, intra- and extranets. To explain that, we have to contrast the virtual internet concept to its conceptual cousins: SDN and NFV.

SDN stands for software defined networks, NFV for network function virtualization. We call applications that use SDN, SDNapp's. Currently SDNapp's, e.g., using OpenFlow [24] technology, may control individual switches, setup and interact with physical Layer 2 paths. NFV implements internet routers, home gateways, mobile base stations and other network equipment. In the context of the Telco industry, NFV is implemented by software running in VMs. Hence, an SDNapp might connect a network link, carrying Layer 3 (Internet) traffic to a virtualized (NFV) router. The current direction of SDN and NFV development is to improve network management, promising ISPs, of course, lower cost of ownership.

The availability of private and public global SDN services is very limited. Even science networks, such as Internet 2 and ESNET, started only recently and on a limited scale to feature SDN services. Indeed, SDN and NFV are positioned as a technology for network owners. This all makes SDN practically unsuitable to be used by many developers of distributed applications. The difficulty to develop and deploy SDNapps contrasts to our approach, where Netapps are the software that operates global scale virtual internets. Only two or three PhD students, and an Amazon grant, were enough to develop the proof of concepts for Netapps that create globe spanning, adaptive, secure, robust and well performing virtual internets, suitable to serve many distributed applications. For our approach to generate a distributed networking infrastructure, thousands of cloud data centers are available globally. The

use of their resources is straightforward and can be arranged via the Internet, e.g., via web services.

This Chapter demonstrates how a virtual internet link, an IP-tunnel over the internet, attains its optimal network topology for a given set of cloud data centers. The publication in this Chapter shows that the QoS, in this case round trip times, of data center based overlay networks frequently outperform the standard internet connection. More advanced optimization algorithms may instantiate multiple paths between source and destination if that improves and matches the quality of service the application needs.

The trial and error optimization method presented in this Chapter is generic and allows the creation of adaptive networks that also optimize other qualities besides latency, e.g., jitter and robustness. Netapps can be constructed that optimize other aspects of the system, such as application specific layer 3 protocols, fragmentation and reassembly, encryption, etc.

This Chapter focuses on the optimization of QoS of the network. In addition to such network oriented optimization, one can also optimize, via distribution and scaling, parts of the application itself. In Chapters 5 and 6 we take run-time distributed application qualities into account and explain the concept of an adaptive virtual infrastructure.

This Chapter, in combination with Chapter 4, shows that a virtual internet is a well performing, software generated and possibly globe spanning internet subnet. One can envision that a future telecommunication, internet and distributed application infrastructure consists of end systems, a finely distributed set of data centers and telecommunication links that interconnect these. Networks are formed in the cloud, packet switching and routing is done in the cloud, most applications run in the cloud. Most application and their internet connections will run in the cloud. Hence, the virtual internet subnets will exceed the number of real subnets in the Internet. The Internet is pushed into the cloud.

3.1 INTRODUCTION

Route selection in the Internet is a consequence of hand-written business-inspired policies. These policies are optimized for and reflect local interests of an autonomous system (AS). However, implementation can be erroneous, leading to sub-optimal end-to-end connections [25]. Since there are almost no end-to-end guaranties for Internet, packets may not travel the shortest path to its destination. As a result, applications do not necessary obtain the optimal end-to-end performance of the Internet.

Applications can influence their own traffic via source routing protocols [26]. However, AS'es usually run protocols like MPLS [27] which work only in a single domain. Recently, several initiatives were launched to create programmable networks [28, 1]. Network switches that support OpenFlow[24] technologies can allow computer programs to enforce their own forwarding rules. However, only a few network

operators employ such switches, and so the control over the routing decisions remains with the operators. Furthermore, the Internet topology and its qualities at a given moment in time may only be partly known [29], making it difficult to find optimal paths. As a consequence, Internet path control is only possible for a part of a network path.

Here, we turn to cloud technology to gain control on packet flows. End-users can allocate virtual resources that can be used as virtual routers. By measuring the quality of the paths between all virtual routers and end-points, we find the optimal path between two end points. If a path with better characteristics is found over a virtual router, we setup tunnels and routing to control the traffic.

The automated management of such overlay networks and their respective cloud resources requires dedicated control structures. Once these control structures are designed and implemented, we need to assess to what extent cloud resources deliver significantly better results with respect to the controllability of network properties.

In this chapter we introduce MeTRO, a framework of **M**anag**e**ment **T**ools for **R**outers **O**n-demand. MeTRO measures and setup routing in a continuous adaptive manner to adjust to change in the Internet. We evaluate our framework in a diverse setup and we find that it can improve on path latency by as much as 95%. MeTRO exhibits such latency improvement behaviour in 58% of the experiments. This reduction means that in substantial cases large scale distributed virtual environments [30, 31] such as; multiplayer online games, distributed military simulation, and collaborative design, become more responsive.

3.2 THE METRO FRAMEWORK

Here, we describe the architecture of MeTRO, our control and measurement framework, and its generic approach to cloud-hosted routers. There are currently two strategies to create optimized paths in the Internet: 1) let ISPs optimize or 2) use methods such as Detour [32]. Since ISPs have an incentive to optimize themselves based on cost, the only viable option is using methods like Detour. Detour adds a fixed intermediate hop between two hosts, to gain (partial) control over the path that a packet takes in the Internet. However, it requires access to physical machines located in different administrative domains.

Building on previous work [32, 33], we present a framework that takes advantage of clouds to control as well as to optimize traffic latency. Intuitively, MeTRO is a framework that finds the optimal path between two endpoints by using VMs residing in the cloud. The main idea is to add one hop (hosted by a VM) between two endpoints and learn whether the given intermediate hop leads to a better path between the two endpoints. Throughout the chapter we use the term

"better path" to describe a path between two points which has the lowest latency among the measured paths between these two points.

MeTRO defines two types of *agents*: 1) *agents* which perform measurements and forward the traffic, and 2) a *controller* which collects the measurement results and adjusts the paths between the agents (by setting-up tunnels and routing). An agent typically runs inside a virtual machine, therefore becoming a *virtual router*; the controller may be deployed anywhere.

Figure 10 shows the closed control-loop implemented by our MeTRO framework. The *measurement-collect-adjust* strategy allows our framework to continuously adapt to the dynamic Internet environment. During the **Setup** stage, the end-user chooses the virtual machine types and locations where to deploy each agent. Next, MeTRO will instantiate the corresponding virtual machines and collect their IPs at the controller. During the **Bootstrap** stage, the controller will copy and remotely execute the script that deploys the agent in each virtual machine. These virtual machines will now act as virtual routers. For allocating and bootstrapping the virtual machines we used Sarastro [2].

Once the virtual routers are running, the MeTRO controller regularly cycles through the following three stages: **Measurement** - executes the latency measurements scripts at every agent; **Collect** - after the measurements are completed, the controller collects all data from every agent, then calculates the fastest paths between all agents; **Adjust** - if a lower latency path is found, MeTRO will build the corresponding overlay network between the endpoints, and setup routing such that traffic is routed accordingly.

The pseudo-code in Algorithm 1 finds the optimal path between two endpoints $\texttt{agent}_a$ and $\texttt{agent}_b$ by using a RTT table and calculating all the possible paths through h agents. This RTT table is populated by the MeTRO controller via the collected measurements. The algorithm is executed by the controller. The `lat` function calculates the latency of the input path, by using the RTT table and taking into account a constant penalty p for each hop. This penalty represents the average time required to forward a packet. The `append` and `remove` functions append and, respectively, remove the second to last element in the list.

Initially, `lst` set to $\texttt{agent}_a$ and $\texttt{agent}_b$. If multiple better paths are found, the path with the lowest latency is picked for setting up an alternative path between the endpoints and dynamically create an overlay network. Currently, we only consider latency for selecting better paths, however, other criteria, such as bandwidth, jitter or a combination of such metrics, can be used to define better paths. The measurements for such network metrics could use tools like Iperf [34] or Nagios [35]. The algorithmic complexity of Algorithm 1 is $\mathcal{O}(n^h)$ where n is the number nodes in the network. While implementing

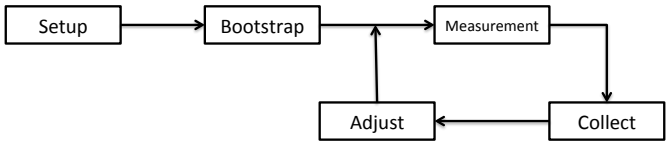


Figure 10: The closed control-loop implemented by the MeTRO framework

Dijkstra's algorithm is more efficient, we choose this algorithm for more clarity.

Require: iplst, lst, h, p, $agent_a$ $agent_b$
  if h == 0 **then**
    return lst
  else
    fastest = lst
    if lat(fastest,p) > lat(direct,p) **then**
      return $agent_a$, $agent_b$
    end if
    for ip ∈ iplst **do**
      if ip ∉ lst **then**
        lst.append(ip)
        t = findLowestPath(iplst, lst, h-1, p, $agent_a$ $agent_b$)
        if lat(t) < lat(fastest) **then**
          fastest = t
        end if
        lst.remove(ip)
      end if
    end for
  end if

Algorithm 1: findLowestPath

3.2.1 *Functional scenarios*

Here, we describe three different scenarios where MeTRO can be employed: 1) troubleshooting routing policies 2) reachability monitoring and 3) virtual networks with specific characteristics.

3.2.1.1 *Troubleshooting routing policies*

Our framework can also be used for finding configuration errors in routing policies. ISPs can determine if their routing policies are optimal by communicating from a given point within their own network to all the measurements nodes. Since clouds offer great flexibility, the ISP can allocate and instantiate multiple VMs around to world by employing our framework. If our measurement framework finds an alternative better path, this means that somewhere in the intermedi-

ate network a routing policy is not optimally configured. A simple *traceroute* from all nodes identifies the node where the policy is not optimal.

Clouds provide a flexible alternative to NLNOG Ring[36], as hosts can be created on-demand for troubleshooting and debugging purposes. In addition, the cost of on-demand virtual machines for collecting views are marginal as opposed to continuously running and maintaining servers. Collecting views using *traceroute* gives insight in how traffic is routed. If a configuration error is made, this will be detectable by comparing multiple views next to each other.

3.2.1.2 *Reachability monitoring*

Our proposed measurement framework can also be used to monitor the reachability of a certain destination throughout the Internet. If a destination is unreachable, it can mean that it is unavailable for every one in the Internet. It can also mean that one of the intermediate ISP "blackholes" the traffic or hijacks the prefix. MeTRO can monitor multiple destinations and check whether specific prefixes are reachable either form inside the network, or outside the network.

3.2.1.3 *Virtual networks*

Another way to use our framework is to create an overlay network using clouds. While overlay networks [37, 32] and virtual networks [38] are well studied, our framework can build a virtual network dedicated to optimizing a specific metric, such as latency, bandwidth etc., over multiple ISPs.

Typical beneficiaries of such overlay networks are large scale distributed virtual environments [30, 31], like multi-player online games, distributed military simulation, and collaborative design. As measurement nodes are scattered around the network, dynamic tunnels are built between these measurement nodes, and the network changes dynamically on the basis of the underlying characteristics of the ISPs.

3.3 EXPERIMENTAL EVALUATION

Our goal is to assess whether cloud-hosted virtual routers are a viable solution for improving the controllability of network properties. To that end, we compare a physical network of routers to virtual routers deployed in the cloud by analyzing the quality of alternative paths found using our Algorithm 1.

3.3.1 *Experimental setup*

Figure 11 shows an overview of our experimental setup consisting of both physical and virtual resources. The endpoints of each exper-

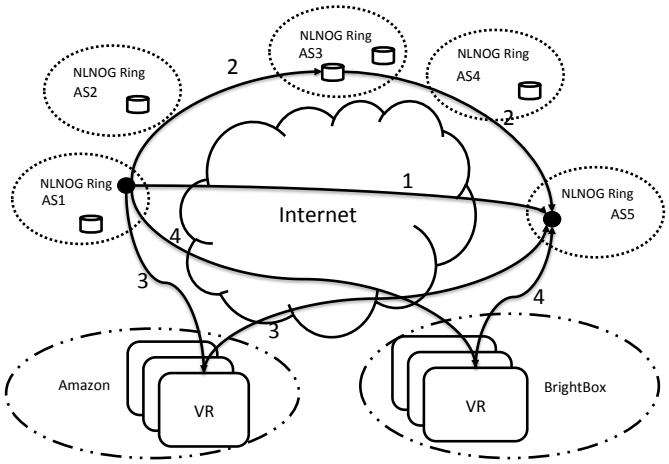


Figure 11: Overview of our experimental setup using NLNOG Ring, Bright Box and Amazon EC2 resources. 1) a direct path between endpoints located in different ASes 2) A path with an additional hop located in NLNOG Ring 3) and 4) paths using a virtual router (VR).

iment are located in a physical network. We have two types of experiments with respect to the intermediate (used for routing) nodes' location: (a) **physical-physical**, where the intermediate node is also located in the physical network, (b) **physical-virtual**, where the intermediate node is located in a cloud resource. For each type of experiment we also look at each Internet protocol considered: IPv4 and IPv6, where available.

The **physical-physical** experiments represent the baseline performance. Here, we performed measurements from every host to every other host (75×74 sets of results). Three round trip time (RTT) measurements were taken every five minutes, over the span of 48 hours, totaling to 129600 measurements from a given node to every other node in the physical network.

For the **physical-virtual** experiments, we selected every possible pair of hosts from the physical network and for each pair we deployed several virtual routers in each cloud zone considered.

To compute the better paths between every two endpoints, we ran our Algorithm 1 with the following parameters: the additional hop penalty $p = 0.3ms$ and the maximum number of hops $h = 8$. The algorithm uses the measurements collected from both the **physical-physical** and **physical-virtual** experiments as input. Hops considered in each experiment are in a single environment, i.e., either NLNOG Ring, BightBox or Amazon EC2.

As a physical network, we used the NLNOG Ring[36], which is an open initiative where network operators donate servers to create network debugging and monitor facilities for their networks. Each participating ISP provides a network measurement node which allows other ISPs to perform measurement and obtain addition information

about the Inetnet routing topology. At the time of our experiments the NLNOG Ring consist of 75 nodes in 72 different Autonomous Systems. Servers in the ring are located in Europe (68), but also in North America (3), Asia (2), Pacific (1) and Africa (1). All the servers of the NLNOG Ring have IPv4 and IPv6 global reachable addresses.

For our **physical-virtual** experiments we used two different cloud providers: Amazon EC2[39] and Bright Box[40]. Amazon EC2 has 10 regions, each with several availability zones. The regions are located in North America (4), South America (1), Europe (1), Asia&Pacific (4). Bright Box has one region with 2 availability zones, both being located in Manchester(GB). Amazon EC2 only provides IPv4 connectivity, while Bright Box provides both IPv4 and IPv6 connectivity.

3.3.2 *Results and discussion*

All results are collected in Table 1, and Figures 12 to 16. Each of them highlights different aspects of our results and we proceed to discuss each of them in more detail in the following.

#(Number of paths)	NLNRv4	NLNRv6	BBv4	BBv6	AMv4
Direct path Faster	2324	2405	n.a.	n.a.	n.a.
1 hop	1050	1078	515	497	1107
2 hops	1143	696	371	243	2921
3 hops	608	473			9273
4 hops	264	392			23944
5 hops	133	230			
6 hops	24	88			
7 hops	4	38			
8 hops	0	2			

Table 1: The total number of "better paths" using a different number of hops located either in the NLNOG Ring (NLNR), BrightBox(BB) or Amazon EC2(AM), for each IP(*v4) and IP(*v6).

Table 1 shows the number of better paths found for each experiment type and considering $h = 0, 1, \ldots, 8$ hops. We notice that the number of better paths in the NLNOG Ring is inversely proportional to the allowed number of hops (h), while the number of better paths in Amazon is directly proportional to h. This is due to the fact that Amazon has multiple zones located geographically close to each other which results in a factorial explosion of the number of paths per region, while the latency offered by the zones in the same region is quite similar.

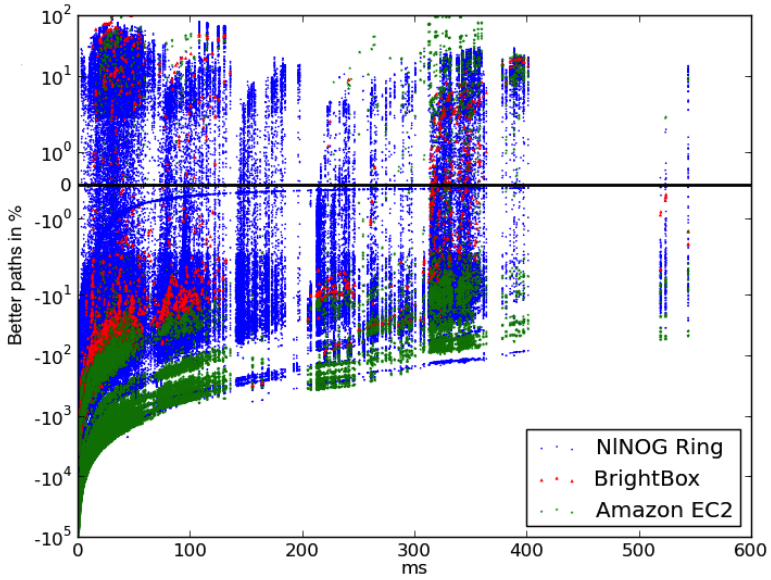


Figure 12: All alternative paths using one hop for IPv4

Figures 12 and 13 plots in log-normal scale all alternative paths found between every pair of hosts. The x-axis values represent the latency of the direct path between each pair of hosts, while each dot represents the percentage by which the alternative path is faster. Negative percentages represent slower paths. Clearly, there are better paths by using an extra hop in a cloud data center. For both IPv4 and IPv6 we can find paths that provide 99% latency improvement for a given source destination pair. On average we find, that if there is a path with low latency, it is 27.53 % better than the original path for IPv4. For IPv6 we see an average improvement of 29.69%.

Figures 14 and 15 show how many better paths are available for each latency interval (40ms). By grouping the number of alternative better paths, we can get an indication of where optimizations of paths latency can be performed. For both IPv4 and IPv6 we see two intervals in which more optimized paths are available: 0 till 200 and 320 till 400. It is more likely to optimize the path if the original latency lies within either interval. The same intervals are there for both Brightbox and Amazon EC2 (IPv4)

Figure 16 shows the NLNOG Ring nodes that provide better paths. When resolving[1] the IPs of these nodes, we found that the best performing nodes are located physically close to large internet exchanges (e.g. AMS-IX, NYIX, DE-CIS, and LINX).

1 IP resolved using GeoIP http://www.maxmind.com

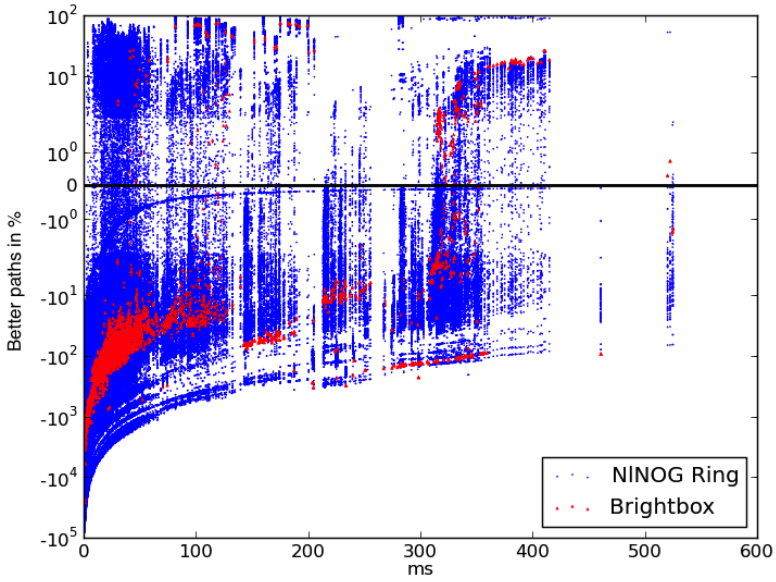


Figure 13: All alternative paths using one hop for IPv6

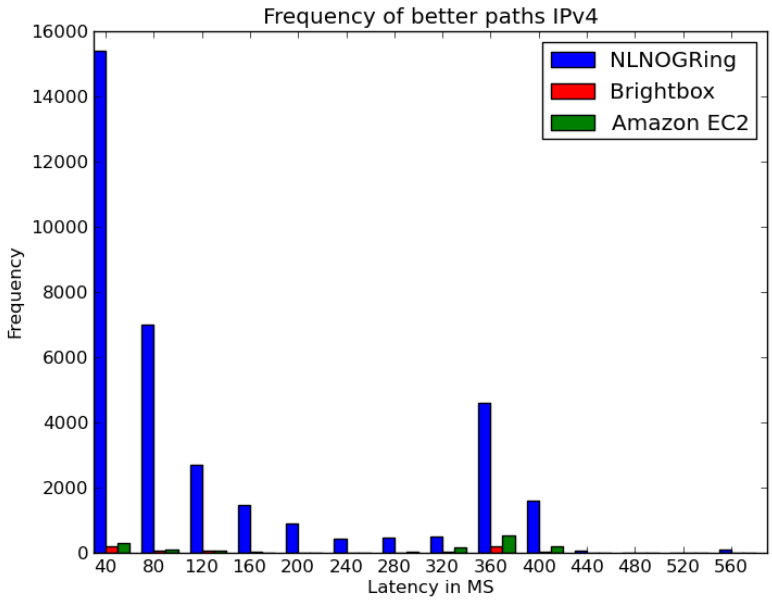


Figure 14: Frequency of "better paths" compared to the direct paths IPv4

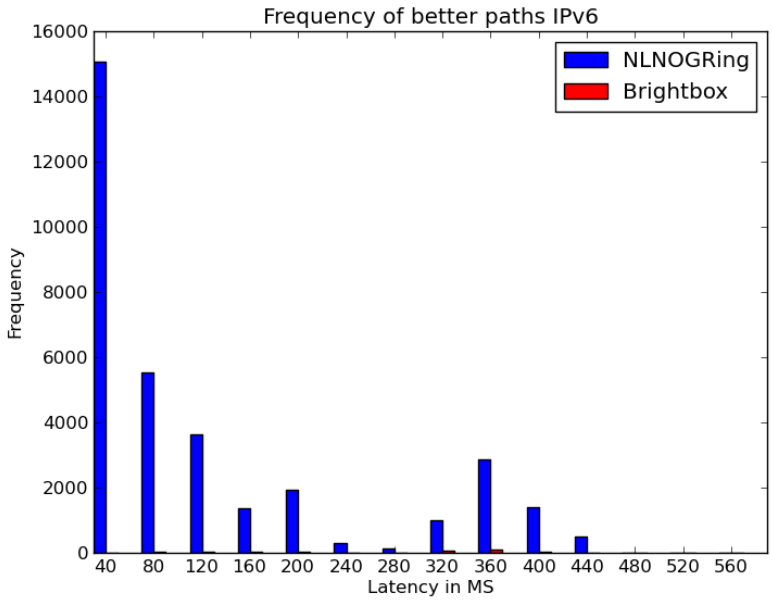


Figure 15: Frequency of "better paths" compared to the direct paths IPv6

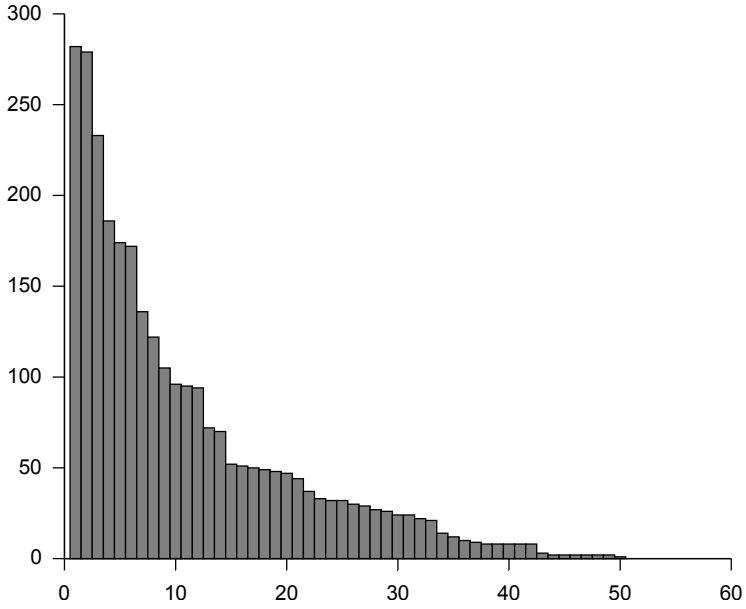


Figure 16: Some nodes provide better paths than others

3.4 RELATED WORK

Savage et al. [32] have shown that even in situations with partial control and knowledge of the Internet topology, alternative paths can be created that have better qualities than those constructed with the current Internet prevailing algorithms. With a technology called Detour,

Savage measured the performance of alternative paths that run via intermediate nodes that were coupled to the Internet at various locations. Ly et al. [33] implemented a similar technology using *traceroute* traces that confirmed the observations of Savage et al. The drawback of both Detour and Ly's approaches is that it requires access to the physical machines representing the intermediate nodes. Our contribution is that we use multiple clouds and an automated framework to find and create better paths in the Internet.

3.5 CONCLUSION

As the Internet optimization domain is partitioned due to the use of BGP, managing static routing policies that are applied to a substantial amount of prefixes in a dynamic environment is left to individual ISP economic priorities and prone to human error. These routing policy errors often lead to bottlenecks, increased latency or traffic can be black holed. To avoid these bottlenecks, users need control and programmability in the network.

In this chapter we introduced MeTRO, a framework of tools that uses cloud infrastructure to deploy alternative, virtual routers. Within this framework, we proposed a method to explore alternative paths with specific properties, such as low latency paths. Our results are very encouraging showing that MeTRO decreases the latency in 58% of the cases studied, albeit increasing the number of hops. We showed that our framework can be used to detect unoptimized routing policies in the Internet, monitor the reachability of the Internet and create virtual networks with specific characteristics.

4

REMOVING THE MEMORY WALL IN SOFTWARE ROUTERS

This Chapter is based on "Fast Packet forwarding engine based on software circuits"[41].

The introduction of Chapter 3 sketched a picture of the future internet. The future internet comprises telecommunication links that carry IP traffic, which is the present day situation, as well as virtual internet links. Virtual internet links are implemented as IP-tunnels between virtual machines that run software routers and applications. Chapter 3 prognotized, on basis of the versatility and security of virtual internets, that the number of virtual internets will exceed the number of regular internet subnetworks. Hence, Chapter 3 forecasted that internet will be pushed into the cloud. To support such hypotheses, one must establish that the performance of the virtual internet infrastructure is at least comparable as implementation alternatives – dedicated Internet routers. The previous chapter showed that latency optimized virtual internets outperformed regular internet connection in more than half of the cases. But what about the performance of the virtual internet routers themselves? The benchmark is set by high end routers that frequently deploy special hardware for packet forwarding, in particular routing table lookup. This was the inspiration for the work presented in this chapter. This Chapter describes the methodology to confine the routing table lookup algorithm to the CPU, thereby minimizing memory interaction. As this memory interaction stalls the processing activity in the CPU, this stalling effect is called the memory wall.

One can identify two implementation extremes of Internet routers. Internet backbone routers, designed to achieve a high performance, frequently use specialized chips to optimize the packet forwarding capacity. Cheaper PC hardware frequently hosts the large number of software routers near the edge of the network. The better network cards in such PCs, offload the CPU and perform many packet operations, e.g., layer 3 checksum computing, very efficiently. Yet, the routing process is predominantly implemented in software. Packet forwarding algorithms for such software routers have been well studied. Traditional algorithms rely heavily on memory loading operations. A common issue with these routing table lookup algorithms, is that memory operations form a bottleneck. The memory bus operates at a fraction of the processors clock-frequency. For every memory load operation, a CPU can execute more than 1000 instructions. The algorithm presented in this chapter avoids fetching data from memory. Instead, it uses four basic instructions: `AND`, `OR`, `XOR` *and* `NOT`. *With only these instructions, the algorithm computes the destination addresses thereby avoiding memory operations. As a result, we achieve a factor ~10 performance gain in routing table lookup on a CPU.*

In case the algorithm is offloaded to a GPU (graphic processing unit), the performance is enhanced substantially further, a factor ~100. This means that the packet forwarding performance of software routers in virtualized networks is comparable to that achieved with internet backbone routers.

There is much relevance of our work for the Telecom industry. Here the softwarization of network elements is known as Network Functions Virtualization, or NFV. Usually, network functions are applications running on VMs. To reach a certain performance level, the number of parallel working virtual machines is scaled. Our work brings another and novel mechanism to enhance the performance of routers in NFV technologies. Combining the results of this chapter with those presented in Chapter 3, we can implement a virtual network link that has better latency and an equal or better throughput as the default Internet path. Furthermore, our fast table lookup method is applicable in many other areas in software engineering.

These engineering achievements can have a large impact. We already stated that a future Internet infrastructure could be based on data centers, connected by telecommunication networks. The data centers host both the virtual networking equipment as well as the applications themselves. The results of the current and previous chapter show that virtual internets perform very well. Moreover, given a finely distributed set of data centers, Netapps can optimize continuously the topology and scaling of the components, i.e., network elements, tunnels, and applications of the virtual internet. Chapter 5 and 6 investigate adaptive distribution and scaling of virtual infrastructure components and application parts.

4.1 INTRODUCTION

The idea of using general purpose computing technologies to replace dedicated, expensive router architectures is not new. Different implementations of packet processing and forwarding for commodity hardware already existed since the early 2000s [42, 43, 44, 45]. For example, in [46], Han *et.al.* presented *Packet Shader*, a high-performance software router framework for general packet processing that can achieve up to 39 GByte/sec on modern CPU/GPUs heterogeneous systems. Their results point out two interesting conclusions: (1) routing using architectures with discrete GPUs suffers from a CPU-GPU communication bottleneck[1], and (2) the performance of the forwarding itself is limited by the internal bandwidth of the GPU.

While the first problem can be addressed by replacing discrete GPUs with fused architectures [47, 48, 49], the latter is an inherent problem of the way forwarding is implemented. Specifically, to make a *forwarding decision*, *i.e.,* to choose to which interface or next hop a packet should be forwarded, traditional algorithms use several memory lookups [50] for each packet. When using multi-core or many-

1 A discrete GPU platform features a CPU (host) and one or several GPUs (devices) connected on a PCI/e bus.

core architectures (such as modern CPUs or GPUs), these algorithms are applied in parallel for multiple packets, potentially leading to a significant improvement in the throughput of the system.

Intuitively, the number of packets to be analyzed and forwarded in parallel should be limited by the number of cores that perform the analysis; for regular GPUs, one can have as many as hundreds of such cores. However, in practice, the number of decisions that can be taken in parallel is limited by the available memory bandwidth of the system: if every unit accesses memory at the same time, the memory bandwidth is quickly saturated, and the amount of concurrency in the system decreases. This performance penalty aggravates when multiple memory accesses are needed. Naturally, the overall throughput of the forwarding process will decrease as well.

Even so, modern multi-core CPUs *and* GPUs remain attractive for implementing such massively parallel forwarding engines because they have very high memory bandwidths - 20–40GB/s for CPUs, and over 300GB/s for GPUs. But their computational throughput far exceeds these memory bandwidth numbers, reaching 100 to 1000 GFLOPs for 8-core CPUs and GPUs, respectively.

In this work, we propose an approach to exploit this extreme computational power. Specifically, we aim at replacing memory lookups with compute operations, thus utilizing much more of the CPU/GPU cores and relieving the memory bandwidth pressure. We base our work on a technique called *bitslicing* [51, 52]. Bitslicing appeared in the early 1960's, proposing to use 1-bit logic chips as building blocks to create multi-bit CPUs. In the context of forwarding decision making, we use bitslicing to create a logical circuit from the *forwarding table* or *forwarding information base* (FIB). The circuit, implemented in software, consists of simple logical operations such as `AND` and `OR`, that take at most 2 cycles on modern architectures. Combined with the parallel forwarding decision-making, which uses the same instruction for multiple packets, this approach is ideal for modern *single instruction multiple data* (SIMD) units (present in CPUs) and *single instruction multiple threads* (SIMT) processors (like GPUs). This approach eliminates most memory lookups, avoiding performance limitations due to memory bandwidth saturation.

In this chapter, we distinguish between *routing*, the process mapping destination networks to output links, and *forwarding*, the process which consults the routing database to decide which output link a single packet should be forwarded on. This chapter focuses on the latter and makes the following contributions:

- We present a packet forwarding engine that uses bit-slicing to concurrently process packets. This approach alleviates the bandwidth saturation limitations of the traditional solutions (Section 4.3).

- We present an algorithm to generate bit-slicing kernels for forwarding IPv4 traffic (Section 4.4) and discuss the requirements, limitations, and potential impact of our bit-slicing approach.
- We provide a qualitative and quantitative evaluation of the advantages and disadvantages of our bitslicing approach. Our results show significant performance advantages for the bit-slicing method, especially for small and medium routing tables (Section 4.5).

4.2 LIMITATIONS AND APPLICABILITY

To handle traffic in a production setting, our approach sacrifices some features which are common in production networks. Some of these limitations imposed by our approach are fundamental, while others are for future work.

Routing updates

In order to accommodate for changes in the forwarding table, the kernel needs to be regenerated with the updated information and go through the OpenCL compile chain. This takes a few milliseconds for the table sizes that we used in our benchmarks. One way to optimize this process is to keep kernels in memory for cases in which a network perturbation is reverted, such that the network is restored to an old kernel readily available. In addition, it is possible to keep a repository of kernels readily generated (from older versions of the routing tables, likely to be restored) and pre-compiled, such that we avoid the overhead of generating and compiling them. We note that the host application does not need restarting when these changes happen, as it is provided with a mechanism that detects such changes, generates and loads the kernel immediately, and can restart processing (pre-buffered) traffic within tens of milliseconds.

Another approach to accommodate with routing updates is a hybrid approach, using both this approach and a traditional memory approach. The complexity of updating routing tables of memory lookup approaches are typical $O(1)$, thus cheaper to this approach. When confronted with a routing update, first update the memory lookup routing table, switch from bit-slicing to memory lookup, and generate a new OpenCL kernel including route update. Finally, switch back to the bit-slicing approach using the new kernel.

Limitations and potential

Our method for forwarding packets works with batches of addresses. In some cases, batch-processing of the IP packets is not applicable -

for example, when traffic is scarce. In such cases, waiting for a batch to be complete might take seconds. A simple solution is to set a time threshold to start processing the current packets in a traditional manner, or pad an incomplete buffer with additional NOP addresses and use the bitsliced engine. Traditional routing can be done, in this case, by frameworks such as Hermes or PacketShader.

Applicability

While our implementation and benchmarks mainly focus on forwarding IPv4, especially looking at destination addresses, the technique presented here can be applied to a much broader spectrum of applications. Examples include routing on specific IPv4 packets flags such as TTL, the option field, (port based) firewalling. Moreover, our technique is suitable for other routing and switching protocols such as MPLS (i.e., 20-bit header) and VLANs (*i.e.,* 16-bit header). In the case of IPv6, for example, it takes not 1, but 7 memory lookups [46] to determine the destination of a given packet. This gives our bitsliced engine the opportunity to gain even more performance compared to traditional memory lookup engines. In principle, we are confident that the technique presented here can be easily applied to new protocols with little to no modification.

4.3 DESIGN AND IMPLEMENTATION

In this section, we discuss memory lookup forwarding algorithms and their limitations. We further introduce the *bitslicing* technique, and show how it can be applied to make forwarding decisions for IPv4.

4.3.1 *Packet processing*

Modern packet processing pipelines, such as those in PacketShader or Hermes, are composed of at least six stages : (1) the packet enters the system via the *network interface card* (NIC), (2) it is copied to main memory; (3) the CPU fetches the packet's header (4) lookup of packet destination in the table stored in the main memory; (5) the packet is modified if needed (*e.g.,* the MAC address is changed), (6) forwarded to the correct NIC transmit buffer.

In this pipeline, an important performance bottleneck is the forwarding decision making, *i.e.,* determining where should packets go next [53, 54]. Recent research [54, 55, 46] has shown that GPUs provide a competitive alternative for implementing forward decision making in stage (4). The basic idea of using accelerators (GPUs or Intel's Xeon Phi) for local forwarding provides *separation of concerns*: the role of the CPU is to receive and transmit packets through NICs (as in

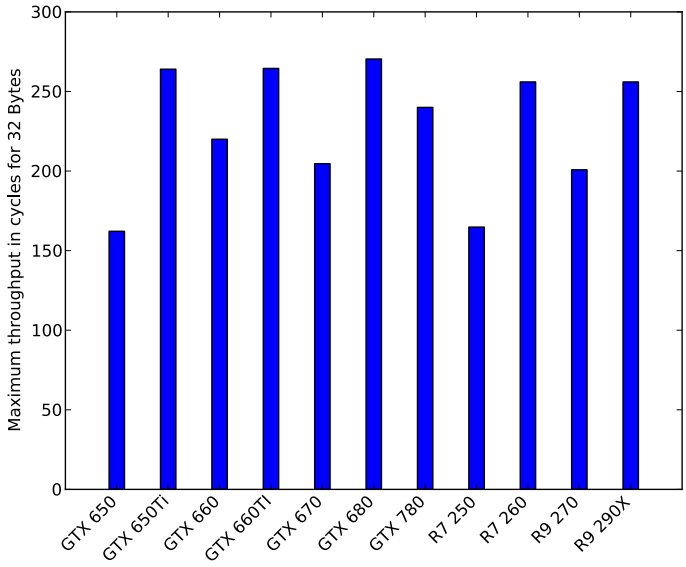


Figure 17: The maximum memory throughput (measured in 32-bytes loads) per core per cycle for NVIDIA (GTX*) and AMD (R*) GPUs.

traditional software routers), while the forwarding decision making is offloaded to the accelerators that (typically) have a significantly higher memory bandwidth.

Our work aims to improve the performance of this particular stage by providing a new implementation of the decision making algorithm and making better use of both the CPUs and GPUs. As we focus on forward decision making only, we assume that the header information is already transferred from the NIC to the main memory being directly accessible to either the processing unit - be it a CPU or a GPU. Note that this assumption does hide a significant performance penalty (due to the data transfer) for the typical discrete CPU-GPU architectures. For such platforms, the additional data transfers cancel the performance benefits of the accelerated decision making. However, existing fused architectures where the CPU and GPU[47] share common memory spaces, do not require additional data copies. Such systems, which are much more suitable for networking devices, can assume data is available in the common memory as soon as stage (1) is finished. Moreover, these devices are programmable using OpenCL, so our implementation should work out-of-the-box for such platforms. However, due to unavailable tools (Linux drivers, compilers, and even hardware), we were unable to properly test these fused architectures yet.

Forwarding algorithms are traditionally based on memory lookups and are limited in performance by the low arithmetic intensity[56] of the processing, regardless of the type of CPU or GPU that is used (despite the 10× difference in bandwidth between the two). Most of

the time the GPU is idle, waiting for data. This is due to two aspects of current architectures 1) the large memory latencies for non-cached data 2) the maximum theoretical bandwidth is shared among cores. Figure 17 shows the minimum number of processing cycles required to receive a 32-byte packet. The number is calculated as follows (pu_speed is the speed in processing cycles per second, $cores$ is the number of cores on the chip, and $max\_memory\_bandwidth$ are all specified by the manufacturer):

$$ratio = \frac{pu\_speed \times \#cores}{max\_memory\_bandwidth}$$

The value lies, for most modern architectures, between 160 and 280 cycles. The memory bandwidth can only be achieved in ideal conditions, where memory accesses are coalesced. This requirement is not met for the case of packet forwarding decision, as no correlation can be assumed between the packets that need to be forwarded. Therefore, these numbers can be even 2–4 times larger. For GPUs that have caches, these numbers are lower, but only in the case temporal locality is available in the networking traffic.

In summary, memory lookups are expensive in both theory and practice. As most arithmetic operations are now done within 1–2 cycles, replacing memory lookups with a (larger) mix of arithmetical instructions can become beneficial, performance-wise.

4.3.2 *Bit-slicing implementation*

Bit-slicing is an old technique (early 1960's, EDSAC 2 computer [57, 51]), where identical small processing units (typically, 1-bit) can be interconnected to make up a larger n-bit processor. Today, bitslicing is mainly used for high speed software applications such as cryptosystems [58, 52]. In this work, we attempt to leverage bitslicing to replace the memory lookups in the forward decision making with computation. For determining where to forward a packet, our proposed method consists of the following 3 steps: 1) translate the forwarding table to Boolean software circuit and create an OpenCL kernel, 2) batch incoming IP destination headers together and bittranspose them, and 3) evaluate the boolean circuit to compute the destination. The remainder of this section explains this procedure in detail.

A packet forwarding decision is essentially the following mapping:

$$\rho : \alpha \mapsto \beta$$

ρ is the routing table, α is the destination address, and β is the next hop in the routed path. In IPv4 routing, a network mask is used to determine the network part of the address, and used to forward the packet to the next hop. In the remainder we assume that network masks are at most 24 bits for IPv4.

For bitslicing to work, we need to translate the routing table to an incomplete truth table which includes the destination addresses. The truth table is then translated to logic instructions (i.e., `AND`/$\wedge$, `XOR`/$\oplus$, etc.) to set the destination address. For example, for the IP address 192.0.0.0/4, where only the first 4 (i.e., 1100) bits are used for forwarding, we compute the following logic expression e: $b_0 \wedge b_1 \wedge \neg b_2 \wedge \neg b_3$. Evaluating this expression is equivalent to executing 3 `AND` operations and 2 negations. We use negations ($\neg$) to match bits that are zero. To exploit the temporal expansion of bitslicing, we require that n IP destination addresses are batched together and that they are bit transposed. This results in the storage of all the first, second, etc. bits of all n IP addresses in the same n-width CPU/GPU register.

Executing the logical expression is done in parallel and all n IP-addresses are computed and available at the same time. The results will be stored in the destination registers, and further bittransposed for a correct final result. The latency of the forwarding decision engine will be given by the number of operations required for 1-bit computation (essentially, the length of the boolean expression), and its throughput will be n.

The translation from the incomplete truth table to logical expressions is straightforward. Each next hop is translated to a unique interface index β. To decide where a packet should go, each logical expression that contributes to a bit in β is evaluated for that bit. The forwarding decision is stored in a array of at most $\lceil \log_2(\max(\beta)) \rceil$.

If the destination is not in the routing table, the logical expressions will not contribute any bit, and the result will be 0. Note that specific behaviour can be assigned to $\beta = 0$ (e.g. dropping a packet or use a default route). Routing packets in the Internet is done based on the most specific prefix, *i.e.*, the expression that matches most bits of the prefix. For the case we have 2 prefixes of different lengths that point to different destinations, we change the Boolean expression such that contributing bits of matching expression are set correctly.

Having the routing table transformed into Boolean circuits, we translate it into an OpenCL kernel [59]. Note that additional boolean optimizations can be applied to the incomplete truth table (e.g., using the Quine-McCluskey algorithm[60]). We chose not to apply such optimizations to limit the overhead of the kernel generation. However, in case the performance results of the kernel are not satisfactory (which has never been the case for our experiments so far), these optimizations can be easily applied, trading-off kernel generation overhead for kernel performance.

4.4 KERNEL GENERATION

Because our implementation is done in OpenCL, we have two types of code 1) *host code*, responsible for initializing buffers, compiling and

launching the device code, and transferring data from host to device memory, and 2) *device* code also known as a *kernel*.

```
Data: routingTable, numNexthops, ips
Result: kernel
for i = 0; i < length(routingTable) do
    s = "unsigned int p" + i + " = "
    for j=0; j < length(routingTable[i]); j++ do
        if j > 0 then
            s = " & "
        end
        if routingTable[i][o][j] == 0 then
            s += " ~ips[id + " + i + " ] "
        end
        else
            s += " ips[id + " + i + " ] "
        end
        kernel += s
    end
end
nh_max = ⌈log2(numNexthops)⌉
for k = 0; k < length(nh_max) do
    s = ""
    for  l = 0; l < length(routingTable) do
        if (1<<k) & routingTable[l][1] == 1 then
            if check_lmp(l, k) then
                s += lmp(l,k,s)
            end
            else
                if isEmptyString(s) then
                    s = " p" + l
                end
                else
                    s += " || p" + l
                end
            end
        end
    end
    if not isEmptryString(s) then
        kernel += "result[rid + " + k + "] = " + s + ";"
    end
end
```

Algorithm 2: Kernel generation algorithm

To make forwarding decisions for packets we must generate kernels that reflect the routing table. Each change in the routing table requires a new kernel to be generated. Algorithm 2 is a pseudo code description of the kernel generation algorithm, which builds the

kernel as a collection of strings. The kernel is compiled and eventually launched by the host. The inputs of the algorithm are the `routingTable`, `numNexthops`, and `ips`. The `routingTable` holds the list of the prefixes (index 0) with corresponding nexthop (index 1). The `ips` is an array that holds the binary transposed destination ip-address. First, every prefix is translated to the binary representation. Every bit in the prefix, either 1 or 0 (which is negated, *i.e.*, "~") is expressed with the logical and operator "&". In addition, the `id` and `rid` are filled in by the global work-item id, for `id` multiplied by 24 as we have prefixes of at most 24 bits, and `rid` multiplied by $\lceil \log_2(\texttt{numNexthops}) \rceil$, such that every thread is working on its own working item.

When the generated statements are executed, every expression is evaluated and stored in a single p_i. The second part of the algorithm is to determine which p_i contributes to which nexthop. For this we match the nexthop index number against the binary index for the array. If the expression matches, it will logically *or* the expression. Prior to checking if an expression contributes to an interface it will try to see if the expression competes for the longest matching prefix as described in Section 4.3.2. The algorithm described here will return only the core of the kernel, the only part that is needed extra are the function statements and the setting of variables `id` and `rid`, as can be seen in Figure 18.

4.5 EVALUATION

In this section we present a thorough evaluation of our method. Therefore, we analyze the performance of our bitsliced engine on multiple GPU and CPU platforms, and we compare it against a state-of-the art algorithm (based on memory lookups) running on the same platforms.

Table 2: A high-level comparison of the tested architectures. The names in this table will be used in all performance graphs presented in this section.

Name	Architecture	#cores	Throughput	Bandwidth
GTX480	GPU, Fermi	480	> 1345GFLOP	~177GB/s
C2050	GPU, Fermi	448	> 1228 GFLOP	~60 GB/s
K20	GPU, Kepler	2496	> 3524.35 GFLOP	~208 GB/s
SB (Intel Xeon E5-2630)	CPU, SandyBridge	12	~ 384 GFLOPs	~42.6 GB/s
6C (Intel Xeon X5650)	CPU	12	~306 GFLOPs	~ 32 GB/s

4.5.1 *Experimental setup*

DIR24 is an IPv4-specific algorithm that uses an array of 2^{24} entries (indexed by 24 bits keys) to store the forwarding information. In our

```
__kernel void
bitslice(__global unsigned int *i, __global unsigned int *r)
{
 int id = 24 * get_global_id(0);
 int rid = 2 * get_global_id(0);

 // 2.104.63.0/14 => 00000010011010, if = 1
 unsigned int p0 = (~i[id + 0 ]) & (~i[id + 1 ]) & (~i[id + 2 ])
     & (~i[id + 3 ]) & (~i[id + 4 ]) & (~i[id + 5 ]) & i[id + 6] &
      (~i[id + 7 ]) & \\
    (~i[id + 8 ]) & i[id + 9] & i[id + 10] & (~i[id + 11 ]) & i[
        id + 12] & (~i[id + 13 ]);

 // 29.247.95.0/15 => 000111011111011 if = 1
 unsigned int p1 = (~i[id + 0 ]) & (~i[id + 1 ]) & (~i[id + 2 ])
     & i[id + 3] & i[id + 4] & i[id + 5] & (~i[id + 6 ]) & i[id +
     7] & \\
    i[id + 8] & i[id + 9] & i[id + 10] & i[id + 11] & (~i[id + 12
         ]) & i[id + 13] & i[id + 14];

 // 89.176.76.0/20 => 01011001101100000100, if = 3
 unsigned int p2 = (~i[id + 0 ]) & i[id + 1] & (~i[id + 2 ]) & i[
     id + 3] & i[id + 4] & (~i[id + 5 ]) & (~i[id + 6 ]) & i[id +
     7] & i[id + 8] & \\
    (~i[id + 9 ]) & i[id + 10] & i[id + 11] & (~i[id + 12 ]) & (~
        i[id + 13 ]) & (~i[id + 14 ]) & (~i[id + 15 ]) & (~i[id +
        16 ]) & i[id + 17] & \\
    (~i[id + 18 ]) & (~i[id + 19 ]);

 r[rid + 0] = p0 | p1 | p2;
 r[rid + 1] = p2 ;
}
```

Figure 18: Example of a generated kernel for 3 prefixes and 3 nexthops

DIR24 implementation we shift the destination IP address of incoming packets by 8 bits and use the remaining 24 bits as an index for the array. Route prefixes shorter than 24 bits will be expanded in the array. For example, the prefix 123.45.0.0/16 will have $2^{24-16} = 256$ array entries associated with it: from 123.45.0 (which is position 8072448) through 123.45.255 (position 8072703). Using this indexing scheme, we require exactly one memory lookup per IP packet.

We have implemented both the DIR-24 and our bitslice implementation in OpenCL [59]. OpenCL is a functionally portable programming model that allows the execution of the same parallel code on different *families* of devices. We chose OpenCL because we want to verify the ability of *both* modern multi-core CPUs and GPUs to handle forwarding. We have run experiments on five different platforms, all part of DAS4[61]. The CPU and GPU details are presented in Table 2.

To benchmark our implementation, we generate a routing table with P prefixes and I interfaces. Each prefix $p_i \in [0, 2^{24}), i \in [0, P)$ has a subnet mask between 0 and 24 bits, randomly chosen. Each p_i is assigned (for redirection) to an interface, randomly chosen between 0 and I. For the experiments included in this chapter, we choose $P \in \{10, 100, 1000\}$, to mimic three routing tables, and we vary the number of interfaces $I \in \{3, 7, 15, 31, 63\}$. We note that these numbers are representative for the majority of the online routers, especially those for small and medium enterprises.

Finally, to simulate traffic, we generate 2^{26} random destination addresses from the routing table. To simulate that the traffic to be forwarded is random, we introduced a *reuse distance* parameter, r, in our traffic generator. When r is very large, there will be very little repetition between prefixes in the traffic (close-to-random traffic). When r is small, we mimic traffic that is more "directional", exhibiting some degree of temporal locality. To take r into account, we construct the traffic as follows: after r entries that are randomly generated, the same r entries are repeated until we reach maximum 2^{26} entries. In this chapter, we present results with $r \in \{1, 8, 256, 8192, 8192000\}$: where $r = 1$ is the identical traffic case, and $r = 8192000$ is the fully random traffic.

4.5.2 *CPU vs. GPU performance*

We present the performance of both DIR24 and bitslice in Figures 19 for the CPUs, and 20 for the GPUs, aiming to see how different generations of platforms impact the obtained performance. In this experiment we use a very high reuse distance, simulating close-to-random traffic. Finally, we use the same OpenCL implementation,

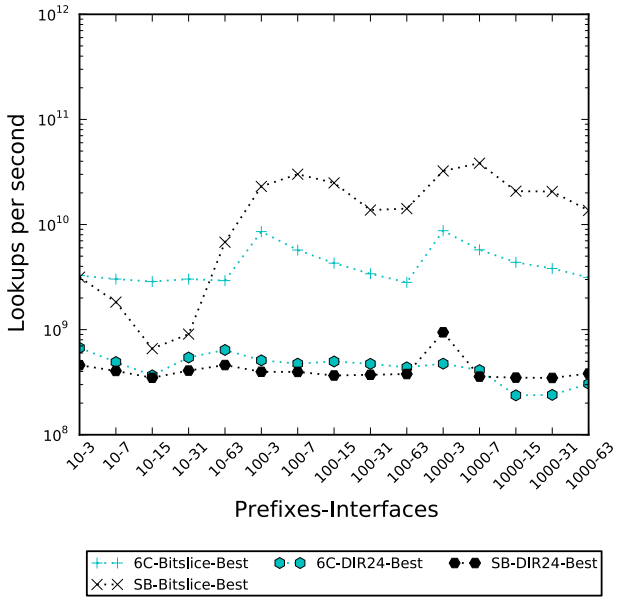


Figure 19: Results for the DIR24 and bitslicing on different CPUs. The horizontal axis specifies P-I - the numbers of prefixes and the number of interfaces in the routing table. The vertical axis shows the execution time in seconds for the different P-I, in log scale. Measurements are taken with reuse distance $r = 8192000$ and workgroup size $wg = 256$.

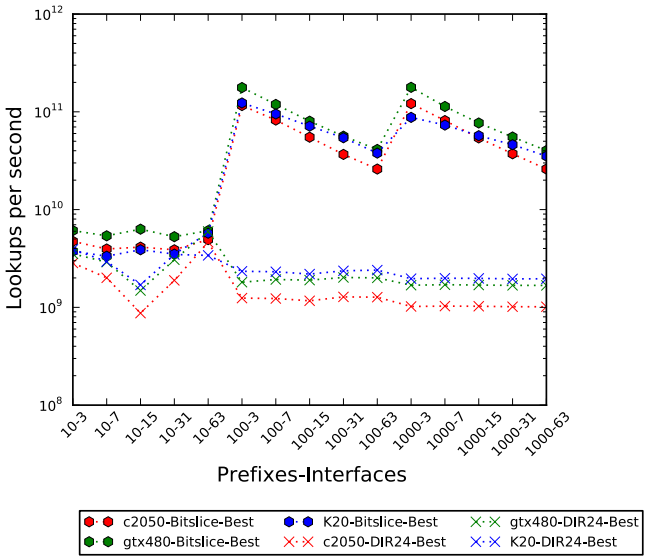


Figure 20: Results for the DIR24 and bitslicing on different GPUs. The horizontal axis specifies P-I - the numbers of prefixes and the number of interfaces in the routing table. The vertical axis shows the execution time in seconds for the different P-I, in log scale. Measurements are taken with reuse distance $r = 8192000$ and workgroup size $wg = 256$.

with a workgroup[2] size of 256 (the following sections will analyze the impact of this choice).

We note that the bitslice version shows higher performance (i.e., lower execution time) for both CPUs and GPUs. For medium sized tables ($P = 100$), the performance gain approaches an order of magnitude. This difference appears because small tables generate relatively simple Boolean circuits, thus requiring only a small number of operations to compute a destination. When the number of prefixes P is larger, more arithmetic instructions are needed, and the bitslice kernel becomes more expensive. Therefore, for $P = 10000$ prefixes, we see the performance of the two approaches converging. Of course, if the number of prefixes grows even further, we expect the DIR24 version to become the better of the two performance-wise, as its performance is practically insensitive to the size of the routing table (hence the relatively straight lines for DIR24 in the figures). The results for $P = 10000$ are omitted in this chapter, as these large tables would only execute sporadically. We expect this is due to the large number of instructions for these tables. For small tables we see only a minor performance increase, the authors suspect that this is a driver nvidia graphics card problem (Cuda version 6.0).

We also note that changing the generation of devices to be used for our forwarding engines does not deliver any significant performance improvement. DIR24 is slightly more sensitive to these changes, probably given the differences in the memory architectures of the systems we tested. Finally, we note that the CPU results show greater variation between different $P - I$ combinations. This behavior is a typical side-effect of using OpenCL on CPUs and the changes this fine-grained model brings to the thread scheduling, the caching behavior, and the auto-vectorization of applications [62, 63, 64].

In addition, the performance comparison between CPU and GPU shows a significant difference between the two solutions, which is somewhat expected given the low parallelism of the kernel: we only process 128 packets[3] per core at the time, which can be dealt with quite well by the SIMD cores of the CPUs we used.

4.5.3 *OpenMP DIR24 implementation*

In a second attempt to boost the performance of DIR24, we have also implemented it using OpenMP, to make sure that potential caching and granularity artifacts that OpenCL introduces are removed. The results of the comparison (CPU-only, as OpenMP can only run on CPUs) are presented in Figure 22. We note that the OpenMP does

2 Threads in OpenCL are grouped in so-called `workgroups`. The size of these groups affects thread scheduling on both the CPU and the GPU, and thus can have performance consequences.

3 This is the width of the SSE unit of the Intel Xeon E5 2630 and X5650

deliver better performance than DIR24 implemented in OpenCL, but it still under performs when compared with the bitsliced engine.

4.5.4 *The impact of caching*

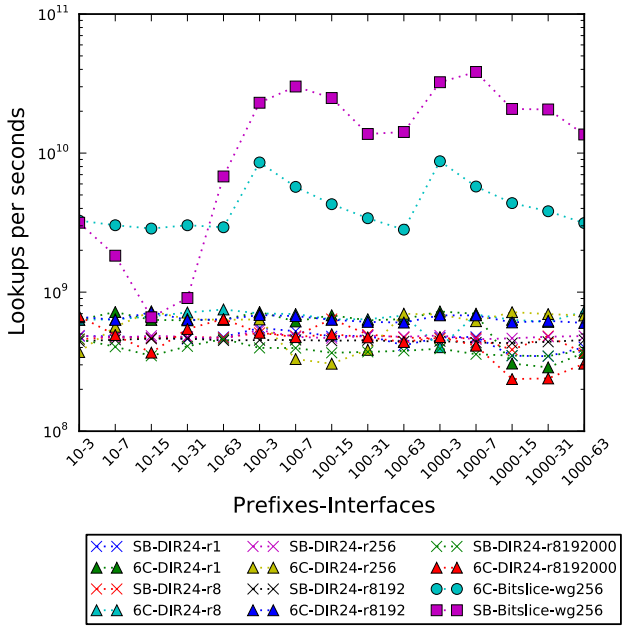


Figure 21: The influence of caching on the DIR24 algorithm for different CPU platforms

In our previous experiments, we have assumed that the traffic that needs to be forwarded is random. We expect, however, that DIR24 is sensitive to different types of traffic, experiencing high performance for smaller r, decreasing as r is increasing. In the implemented DIR24 algorithm, the smaller reuse distance r for the traffic generated mimics the load behavior of hash lookup algorithms[50, 65]. For the bitslice engine, r should have no direct impact on performance, because this implementation uses no memory lookups. The results of varying r for Different CPUs are presented in Figure 21.

We note very little performance improvement in the DIR24 algorithm for the CPU platforms. We believe this effect is due to the negative impact of OpenCL's fine granularity on the effectiveness of caching [62]: due to the scheduling of work items on threads and cores (determined by the implementation and mapping of the model on the architecture - in this case, the Intel SDK), the accesses pattern of neighboring elements becomes unpredictable. As expected, the bitslice shows no significant variation for better or worse caching. For completeness, we ran the same experiment with DIR24 and various reuse distances on the GPU platforms. Similar with the CPUs, our

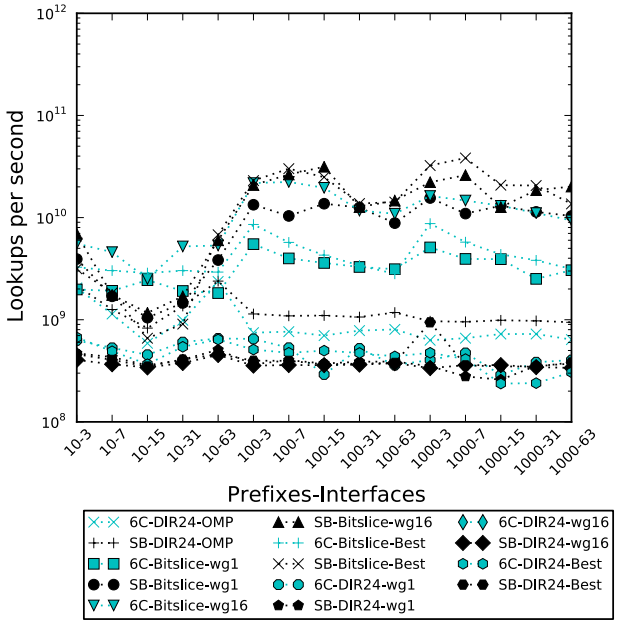


Figure 22: The impact on the performances for different bit-size. The comparison between OpenCL and OpenMP for DIR24 algorithm on different CPU architectures. The results for OpenCL are obtained with $wg = 256$, and the reported OpenMP results are the best ones when varying the number of threads from 2 to 16. The reuse distance is $r = 8192000$.

results show no significant performance variation due to potential improved caching, meaning that our bitsliced engine remains superior to DIR24 even for non-random traffic.

4.6 RELATED WORK

In this section we give a brief overview of related research and position our work in this landscape. IP lookup algorithms have been well studied over the past decades. The very first solutions were software based, but later designs were implemented in dedicated circuits (Application-Specific Integrated Circuits or ASICs) to overcome the mismatch between network and lookup processing speeds.

Traditionally, packet forwarding algorithms use trees to limit the memory usage of the system. In the 1990s, Radix trees [66] dominated the landscape for forwarding packets. To optimize for space, the radix tree data structure merges nodes which have only a single child. To reduce the number of search steps, Nilsson *et.al.* proposed in [67] to compress long paths to speedup forwarding. Another proposed optimization is n-ary branching [68], where multiple small arrays of n elements are used, allowing for better caching.

Some solutions are load-optimized as opposed to space-optimized to gain more performance. One example is DIR-24-8 [53] (referred to further as DIR24), where the almost complete IP space is mapped in memory to the next hop. Similarly, the Lulea scheme [69] exploits the tree like structure with three levels arrays (using multiple tables of hold portions of the IP space) and then uses compact representation of pointers. In the same race to speedup the forwarding, Waldvogel *et.al.* [50] propose the use of separate hash tables for each prefix length, starting the search from the most specific prefix and ending at the least specific one. This scheme is elegant but not particularly fast compared to other solutions for IPv4. Finally, several algorithms exploit on-chip caches, such as those in [65] and [70].

When GPUs emerged as promising bandwidth-rich architectures, GPU-based software routers were proposed. In [46], Han *et.al.* introduced *PacketShader* that demonstrated the potential of using GPUs by showing a throughput of 39.2Gb/sec. *PacketsShader* uses a hybrid approach, where both the CPU and the GPU work to forward traffic. *Hermes* [71], introduced shortly after PacketShader, also used a combination of CPU and GPU routing, and provided the added benefit of Quality of Service (QoS). Both *PacketShader* and *Hermes* separated the forward lookup engine from the packet handling: the packet handling is done on the CPU (avoiding unnecessary data transfers to the GPU), the lookup engine is executed on the GPU.

In 2011 another GPU lookup engine *GALE* [54] has been proved to achieve higher forwarding speeds than PacketShader. *GALE* uses a single table which stores all possible prefixes no longer than 24 bits, resembling the DIR24 algorithm [53]. In 2013, Li *et.al.* introduced a *GPU accelerated multi-bit tree* (GAMT) [55] lookup engine, an engine that explicitly handles forwarding tables updates. GAMT outperforms GALE, but it still uses memory lookups to decide where to forward the packet.

In fact, all the above-mentioned systems and engines are based on memory lookups, *i.e.*, for every packet some information regarding the destination has to be fetched from memory. This approach is ultimately limited by the bandwidth of the system that remains much lower than the computational throughput for many modern architectures. In this work, we present an orthogonal technique: we build an engine that *computes* the destination of a packet by evaluating software-based Boolean circuits in parallel and capitalizes on the much larger computational throughput of modern parallel architectures.

4.7 CONCLUSION

While routing on commodity hardware has been researched extensively, we are trying to improve upon existing approaches by explor-

ing new methods. In this chapter, we explored a new opportunity to provide good performance for the forward decision making in software on commodity hardware. Our new technique, based on bit-slicing, which replaces the traditional memory lookup based scenarios for identifying the destination of a packet with a compute-friendly version. We have implemented this technique in OpenCL, to have quick access to multiple families of platforms, and evaluated it.

Our evaluation indicates that the bit-slicing technique is useful as long as the complexity of the routing table remains manageable. If the number of operations needed to compute the destination is too large (more than 10000), the performance of memory lookups will surpass the performance of bit-slicing. We have also shown evidence that this first implementation outperforms the traditional solutions implemented in both OpenCL and OpenMP on GPUs and CPUs. Therefore, the applicability of our technique makes it ideal for packet forwarding algorithms with small routing tables, such as MPLS or VLAN. Finally, we have shown that that our bit-slicing achieves similar speedups on both CPUs and CPU. Meaning that even on low-end CPU, GPUs and even fused architectures such as many consumer products can benefit from the presented technique.

5

COMPUTING THE CO_2 COST OF GLOBALLY DISTRIBUTED APPLICATIONS

This Chapter is based on "A decision framework for placement of applications in clouds that minimizes their carbon footprint"[72].

The virtual internet concept allows to scale and distribute applications globally, potentially using a significant amount of ICT infrastructure and having a likewise environmental impact. Hence, energy efficiency and minimal CO_2 emission are relevant design issues for virtual internets. The work in this Chapter provides a model that estimates the CO_2 system emissions and allows to create Netapps that minimize the environmental footprint of virtual internets.

The model that computes the CO_2 emissions is quite elaborate and combines results presented in scientific literature. In particular, the CO_2 emissions of both computing and networking elements are included as well as the power efficiency of data centers. Furthermore, the model incorporates the effects of the production process of the electric power (i.e., hydro, brown cole, natural gas, etc.). Although we were not able to verify the model experimentally, it does provide quantitative results where none was available hitherto. Importantly, the method does not need, or reveal, specific details of the application, the applicability is generic. With additional engineering, the model can be provided as "a service" to Netapp designers and forms an "hello world" for scientists who want to improve upon it.

We can combine the results of the previous chapters with this chapter. One can use this chapter to calculate for a given CO_2 budget the number of VMs and how they are distributed over a set of data centers. With the algorithms provided by Chapter 3 one can create optimal performing virtual internet paths between the VMs. With the technology of Chapter 4 one can implement fast software routers. With this chapter one can optimize a VM distribution in such a way that CO_2 cost is acceptable. The next chapter provides a method that, given the VM distribution, optimizes a distributed application on them. Here, NetApps maximize the performance of applications by distributing them most efficiently over available VMs.

5.1 INTRODUCTION

From a user's perspective, reducing the environmental load of the computational tasks is equivalent to looking for a green data center, *i.e.*, a data center with a low power usage effectiveness (PUE). A data center with a low PUE uses it energy more efficient. Many data centers advertise their greenness as an added value for customers. A recent study [73] shows that 71% of the data centers measure the PUE

and that the mean value is about 1.8. Another survey for data centers in Europe [74] came up with a higher mean value of the PUE. Some large data centers claim to have a PUE approaching the theoretical value of 1. We argue that the PUE is not the only factor to consider: the energy sources powering a data center and the network used to move the data are also important, as they determine the amount of CO_2 emitted for a given task.

We will present a framework that facilitates a user to decide where to perform a task, whether at a local data center or remote at a cleaner data center. The framework does not only take the CO_2 emission of the data centers into account, but also estimates the CO_2 emission of the transport network between them when input/output data accompanied the task. We can do this by exploiting the relation between energy produced in kWh and CO_2 emission for different energy sources (see equation 9). The CO_2 emission of the network of a data center is a modest part of the total CO_2 of the data center [75]. However, deciding if offloading of an individual task to an optional cleaner data center is preferable, the contribution of the network (data center LAN and transport network) can be a substantial part of the decision. This means that if the decision framework introduced in this chapter will be applied to all jobs of a data center, the total CO_2 emission of both the data center and the optional cleaner data centers for offloading tasks will decrease.

The framework can make a prediction of the total CO_2 emission for different scenarios, namely software interactive computation and hot or cold data storage. For each scenario we identify the equipment required in the local and the remote data center, e.g., for a computational task other equipment is used than for hot data storage. Subsequently we use models including the power consumption of the devices in use. In this chapter we will focus on the computational scenario, but the interested reader can find some details of the storage scenario in [76, 77].

A common aspect of all scenarios considered is the amount of data involved. The input data determines the energy cost of the data transport part, first through the LAN of the local data center, then across the core network, Internet or light path, and finally through the LAN of the chosen remote data center. When output data plays a role we assume that the user is located near the local data center, so the energy cost associated with the output data is the energy cost for the local LAN versus the cost of the remote LAN and the transport network.

The equipment present in a data center, including the LAN devices, can be more realistically identified than the number of devices in a transport network to another data center. For the former we chose the same internal architecture for both data centers being compared; this allows us to purely focus on the sustainability of both. The latter instead depends not only on the type of network, Internet or light path,

but also on the geographical location of both data centers. Therefore, our framework makes use of network models depending on the type of network and on the location of both endpoints to give an estimate of the minimal number of hops in the network. Furthermore, the geographical location of both endpoints determines the possible countries crossed by the shortest path transport network. These estimates make it possible to attach a CO_2 emission to the transport network. Data on the energy types used by different European countries is available. If the transport network e.g., connects a data center in the Netherlands with one in Austria, a considerable part of the shortest path network will cross Germany. So the energy cost can be divided in three contributions, according to the distance spanned in each of the countries crossed. For each country we can calculate a mean CO_2 emission based on the types of energy sources used in that country [78, 79, 80, 81, 82, 83].

The rules applied to facilitate a user in his decision can also be applied by a scheduler of a data center. If a user can specify the complexity of his task, i.e., how computation time or the amount of output data scale as a function of the input data, a scheduler can determine where to schedule the job such that the emission of the task in gr. CO_2 is minimal. In that case the user need not to know about remote data centers and their PUE's, because this knowledge resides in the scheduler's database.

5.2 RELATED WORK

There are different aspects one can focus on in the optimization process of data center infrastructure costs. We chose to concentrate on CO_2 emission costs, but there are other possible focus points such as economic costs, power utilization and infrastructure utilization. For each one of these costs there is ample existing research: namely for economic costs the work done by [84, 85, 86], for power utilization the work by [85, 86] and [84, 87] for infrastructure utilization.

Optimization of each of these aspects can lead to different outcomes. For example, a data center running more energy efficiently but supplied by energy produced from brown coal has a higher CO_2 emission cost than a data center operating much less efficiently that is using hydroelectric power.

In this chapter we focus on CO_2 emissions costs. What for us is of interest is the ever-increasing effort in modeling the power consumption of networks and data center equipment. Understanding the power consumption in more detail of networks and computer equipment and their behaviour under different conditions, gives the opportunity to better predict the impact of cloud computing and storage on the environment and to develop algorithms and strategies to reduce the carbon footprint. The way we predict the energy consumption of

LAN's and transport networks is based on the work of Baliga et al. [88].

We distinguish different kind of networks, LAN's, Internet and light path, each with their specific type of equipment. Our novel contribution is that we integrate and extend different models into a single decision framework for greener computing. The models used can be easily enhanced, allowing the framework to evolve if one wishes. Our main impetus for the framework presented is that not only end users but also data centers' operators and cloud service providers should think under what conditions it is better to host a job locally, or to host it elsewhere.

5.3 ENERGY MODEL

When deciding to move data and the accompanied computation from a local to a remote data center we have to define an energy consumption metric that accounts for both data centers and the transport network between them. With this metric we should be able to calculate values for the following equation that indicates when movement to a remote data center is to be preferred above local processing:

$$\begin{aligned} &\text{Energy cost local processing} > \\ &\text{Energy cost network} + \text{Energy cost remote processing} \end{aligned} \tag{1}$$

where:

$$\begin{aligned} &\text{Energy cost network} = \\ &\text{Energy cost of local data center LAN} + \\ &\text{Energy cost transport network} + \\ &\text{Energy cost of remote data center LAN} \end{aligned} \tag{2}$$

In the following sections we will focus on two different aspects that contribute to equation 1: how efficient a data center uses its energy, and what are the different components used in the data center and the network.

5.3.1 *How efficiently a data center uses its energy*

To rate the energy efficiency of data centers the commonly used number is the PUE. The PUE is expressed as the ratio of the total power consumption of a data center (P_{TOT}) to the total power consumption of IT equipment like storage devices, servers, routers (P_{IT}).

$$PUE = \frac{P_{TOT}}{P_{IT}}, 1 < PUE < \infty \tag{3}$$

In the calculation of the PUE of a data center all equipment that is not considered a computing device, like pumps, air conditioners, lighting, are part of P_{TOT} only, whereas the power used by servers, storage equipment, network equipment are incorporated in both P_{IT} and P_{TOT}.

5.3.2 *The different data center and network components used*

An important conclusion of a recent study by Tucker [89] is that in a global scale (data) network, the energy consumption of the switching infrastructure is larger than the energy consumption of the transport infrastructure'. We will therefore make a distinction between optical communication systems and conventional Ethernet. We will restrict ourselves to the case where the end user is directly attached to the data center clouds/clusters via a corporate network. The user (or a scheduling application on his behalf) must decide whether the data with the accompanied computation stays at a data center or should be moved to another data center. If he decides to move data, the data will be transported over a public data network given that different data centers are mostly geographically separated. When data traverses the Internet energy consumption can be estimated by adding the contributions to the energy of switches, amplifiers, transceivers, etc. that the data traverses. At both sides, at the local and remote data center, we have the local area network (LAN) of the data center itself that connects the data storage devices and servers to the outside world, i.e., the transport network. To keep calculations simple we assume the same components are present in the LAN of any data center. Table 1 lists the typical equipment data traverses through the LAN of a data center.

Table 3: Components of a data center LAN

LAN data center
Host (network interface)
2× Switch
2× Firewall
Switch
Router

According to Table 1 we arrive (see Baliga et al. [88] Eq. 2) at the following equation for the energy consumption per bit for the LAN of a data center:

$$\hat{E}_{LAN\_data\_center} = \frac{PUE}{U} \cdot \left(\frac{P_{host}}{C_{host}} + 3\frac{P_{switch}}{C_{switch}} + 2\frac{P_{firewall}}{C_{firewall}} + \frac{P_{router}}{C_{router}} \right) [W/bit/s] \quad (4)$$

where $P_{host}, P_{switch}, P_{firewall}$, and P_{router} are the power consumed by the host computer where the data resides, Ethernet switches, firewall, and data center gateway router, respectively. The capacities of the corresponding equipment and measured in bits per second are given by $C_{host}, C_{switch}, C_{firewall}$, and C_{router}.

Here, the factor U accounts for the utilization of the network equipment, expressing the fact network equipment typically does not operate at a full utilization while still consuming 100% of the power [90], a factor we took equals to 0.5.

Data transfers across a transport network can use two different types of connections: the regular Internet and dedicated connections. The regular Internet is available to all users, while in principle dedicated connections (light paths) are more frequently encountered in scientific and corporate environments for high-end users. In both cases the data transfer can be over long or short distances, and we account for this in our model. Figure 23 and 24 show the data network building blocks we assume to be representative for Internet and light path networks.

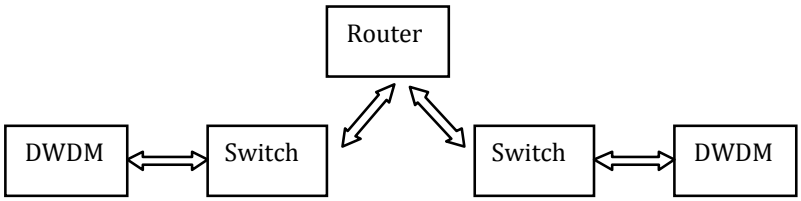


Figure 23: Network components in an Internet building block representing a hop.

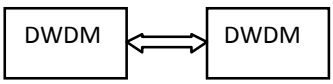


Figure 24: Network components in an light path building block representing a hop.

With these building blocks we compose short and long distance network paths. Multiple Internet building blocks are connected to each other, and multiple light path building blocks are connected via a switch with each other. The entry points and exit points for any kind of data network are a switch connected to a dense wavelength

division multiplexing node (DWDM). Baliga et al. [88] take a mean number of hops for each kind of network (Internet and light path), where we take the number of hops for each kind of network depending on the geographical position of both endpoints. Figure 25, 26, 27, and 28 show the example diagrams for single hop and three hop Internet and light path networks.

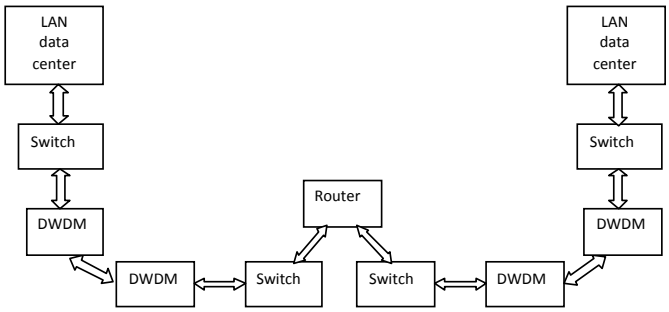


Figure 25: Short distance Internet of 1 hop between two data centers.

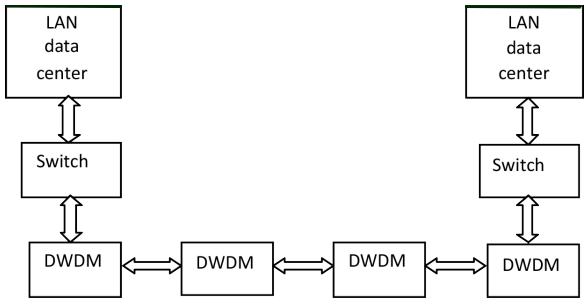


Figure 26: Short distance light path of 1 hop between two data centers.

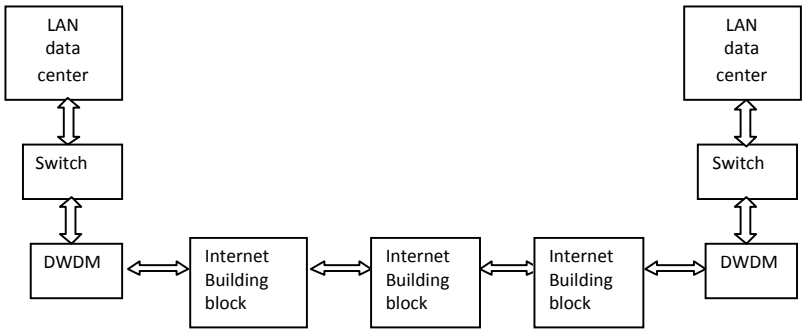


Figure 27: A long distance Internet of 3 hop between two data centers.

We write for the processing cost of a task in equation 1 :

$$E_{processing\_data\_center}(T_{processing}) = PUE_{data\_center} \cdot P_{comp\_host} \cdot T_{processing}\ [kWh] \tag{5}$$

where P_{comp_host} is the power consumption of a computation host in kW and $T_{processing}$ the processing time in CPU core hours. If the

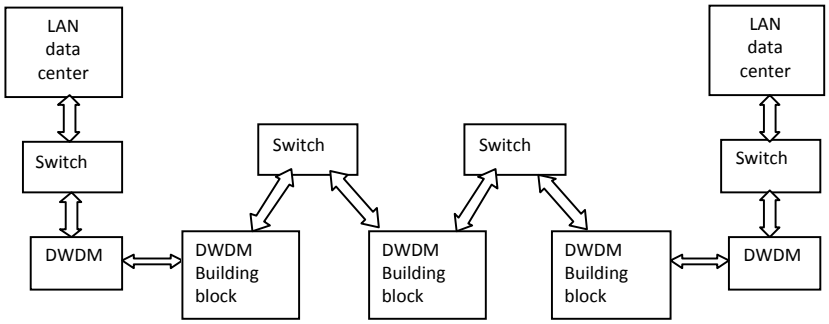


Figure 28: A long distance light path of 3 hop between two data centers.

task is accompanied with N_{in} GByte of input data, this data will always be transferred through the LAN of the local data center. In case the task will be processed at a remote data center, this data will be once more transferred through the LAN of local center, subsequently the connecting transport network and the LAN of the remote data center. The transport cost of the LANs follow from Equation 4.

$$E_{LAN\_data\_center}(N_{in}) = \frac{PUE_{data\_center}}{U} \cdot \left(\frac{P_{host}}{C_{host}} + \frac{3P_{switch}}{C_{switch}} + \frac{2P_{firewall}}{C_{firewall}} + \frac{P_{router}}{C_{router}}\right) \cdot \frac{8N_{in}}{3600}[kWh] \tag{6}$$

while the connecting transport network cost will depend on the type of network, Internet or light path, and the number of hops:

$$E_{transport\_internet}(N_{in}) = \frac{PUE_{network}}{U} \cdot \left(\left(\frac{2P_{switch}}{C_{switch}} + \frac{2P_{DWDM}}{C_{DWDM}}\right) + \left(\frac{2P_{switch}}{C_{switch}} + \frac{2P_{DWDM}}{C_{DWDM}} + \frac{P_{router}}{C_{router}}\right) \cdot n_{hops}\right) \cdot \frac{8N_{in}}{3600}[kWh] \tag{7}$$

$$E_{transport\_lightpath}(N_{in}) = \frac{PUE_{network}}{U} \cdot \left(\left(\frac{2P_{switch}}{C_{switch}} + \frac{2P_{DWDM}}{C_{DWDM}}\right) + \left(\frac{2P_{DWDM}}{C_{DWDM}}\right) \cdot n_{hops} + \left(\frac{P_{switch}}{C_{switch}}\right)(n_{hops} - 1)\right) \cdot \frac{8N_{in}}{3600}[kWh] \tag{8}$$

where the factor 8 accounts for the translation of bytes into bits, as the terms P/C are measured in kW/Gb/s.

In order to solve eq. 1 for the total energy consumption to move data we need values for the different equipment the data traverses.

Table 2 lists the adopted values for the power per capacity (P/C) in kW/Gb/s of the devices listed in Table 1 and depicted in figure 23 and 24. All values are taken from [88] except the value for routers which we obtained from measurements at our local data center.

Table 4: Power per capacity for the different components in our model.

Equipment	Power per capacity [kW/Gb/s]
Host data storage	0.2800
Router	0.0120
Ethernet switch	0.0230
Firewall	0.0160
DWDM terminal node	0.0034

5.4 SUSTAINABILITY

We are interested in the sustainability aspects of the energy sources used in the data network and data centers, and in the subsequent CO_2 emissions. One way we propose to incorporate this, is to transform energy cost in kWh to carbon emission cost effects. A kWh can be converted into grams of produced CO_2 according to the following formula

$$1kWh \sim Xgr.CO_2 \tag{9}$$

where the value of the factor X depends on the type of energy source, *e.g.* $X = 870$ for anthracite electricity production, and $X = 370$ for gas electricity production. In our framework values for X are compiled from different sources [78, 79, 80], leading to the values presented in Table 5.

We can now map the energy costs in kWh given by equations 5, 6, 7 and 8 into an equivalent carbon emission cost K in terms of grams of CO_2 produced:

$$K_{processing\_data\_center}(X_{data\_center}, T_{processing}) = X_{data\_center} \cdot E_{processing\_data\_center}(T_{processing}) \tag{10}$$

$$K_{LAN\_data\_center}(X_{data\_center}, N_{in}) = X_{data\_center} \cdot E_{LAN\_data\_center}(N_{in}) \tag{11}$$

$$K_{transport\_network}(X_{transport\_network}, N_{in}) = X_{transport\_network} \cdot E_{transport\_network}(N_{in}) \tag{12}$$

Table 5: Values for the factor X used in our framework as function of the different energy sources (in decreasing value of X)

Energy source	X value
Lignite/brown coal	950
Anthracite	870
Crude oil	640
Gas works gas	400
Natural gas	380
Nuclear power	66
Geothermal power	40
Biomass	30
Solar power	22
Hydroelectricity	15
Wind power	10

Decision equation 1 for transporting data with accompanied computation to another data center transformed to grams of CO_2 produced now reads:

$$\begin{aligned} K_{processing\_local\_dc}(X_{local\_dc}, T_{processing}) + \\ K_{LAN\_local\_dc}(X_{local\_dc}, N_{in}) > \\ 2 \cdot K_{LAN\_local\_dc}(X_{local\_dc}, N_{in}) + \\ K_{transport\_network}(X_{transport\_network}, N_{in}) + \\ K_{LAN\_remote\_dc}(X_{remote\_dc}, N_{in}) + \\ K_{processing\_remote\_dc}(X_{remote\_dc}, T_{processing}) \end{aligned} \tag{13}$$

The terms on the left of the equation describe the total emission if the computation task is performed locally, while the terms on the right site concern the emission cost if the task is offloaded to and performed at a remote data center. Left we see the contribution of the LAN for the data coming in once, while on the right we see the LAN of the local data center contributes twice, as the data needs to come in from the owner and after the decision is sent out towards the remote data center. In case we have to deal with output data from a computational task we assume that the one interested in the output

data is located near the local data center, and we extend equation 13 to:

$$\begin{aligned} K_{processing\_local\_dc}(X_{local\_dc}, T_{processing}) + \\ K_{LAN\_local\_dc}(X_{local\_dc}, N_{in}) + \\ K_{LAN\_local\_dc}(X_{local\_dc}, N_{out}) > \\ 2 \cdot K_{LAN\_local\_dc}(X_{local\_dc}, N_{in}) + \\ K_{transport\_network}(X_{transport\_network}, N_{in}) + \\ K_{LAN\_remote\_dc}(X_{remote\_dc}, N_{in}) + \\ K_{processing\_remote\_dc}(X_{remote\_dc}, T_{processing}) + \\ K_{LAN\_remote\_dc}(X_{remote\_dc}, N_{out}) + \\ K_{transport\_network}(X_{transport\_dc}, N_{out}) \end{aligned} \tag{14}$$

5.5 DECISION FRAMEWORK

Equations 13 and 14 are at the basis of our decision framework. They can be used in decision policies taken by a scheduler (section 5.1) as well as in a web calculator available to end users (section 5.2). A scheduler will take a decision on where to place computation based on these policies, and it will provide the user with detailed information on the CO_2 emission cost of the chosen scenario. The complexity of tasks, *i.e.*, how the computation time scales with the input data and how the output data scales with the input data, is a factor included in the decision framework too.

5.5.1 *Decision policies*

If a user submits a task and indicates the processing time and the amount of input data needed, and the amount of output data expected, a scheduler should be able to decide whether the task can be better performed locally or at another remote data center from a knowledge base. To decide whether a remote data center is a greener option the scheduler applies equation 14 as a decision policy, which can be written as follows:

$$
\begin{aligned}
&X_{local\_dc} \cdot PUE_{local\_dc} \cdot P_{comp.host\_local\_dc} \cdot T_{processing} + \\
&X_{local\_dc} \cdot PUE_{local\_dc} \cdot E_{LAN\_local\_dc} \cdot N_{in} + \\
&X_{local\_dc} \cdot PUE_{local\_dc} \cdot E_{LAN\_local\_dc} \cdot N_{out} > \\
&2 \cdot X_{local\_dc} \cdot PUE_{local\_dc} \cdot E_{LAN\_local\_dc} \cdot N_{in} + \\
&X_{network} \cdot PUE_{network} \cdot E_{network} \cdot N_{in} + \\
&X_{remote\_dc} \cdot PUE_{remote\_dc} \cdot E_{LAN\_remote\_dc} \cdot N_{in} + \\
&X_{remote\_dc} \cdot PUE_{remote\_dc} \cdot P_{comp.host\_remote\_dc} \cdot T_{processing} + \\
&X_{remote\_dc} \cdot PUE_{remote\_dc} \cdot E_{LAN\_remote\_dc} \cdot N_{out} + \\
&X_{network} \cdot PUE_{network} \cdot E_{network} \cdot N_{out}
\end{aligned}
\tag{15}
$$

where $T_{processing}, N_{in}, N_{out}$ are respectively the computation time in CPU core hours, the amount of input data and the amount of output data, both in GBytes. Furthermore $E_{LAN\_local\_dc}$, $E_{LAN\_remote\_dc}$ and $E_{network}$ are unit energy consumptions of the data center LANs and the connecting transport network, expressed in kWh/GByte. Values for X_{local_dc}, PUE_{local_dc}, X_{remote_dc}, and PUE_{remote_dc} reside in a knowledge base of the scheduler. The values $P_{comp.host\_local\_dc} = P_{comp.host\_remote\_dc} = 0.355kW$ [88] and $E_{LAN\_local\_dc} = E_{LAN\_remote\_dc} = 0.0017kWh/GByte$ (derived from equation 6 with the adopted values for network equipment) are constants for any decision policy, whereas the value for $E_{network}$ depends on the type of network and on the number of different hops, equations 7 and 8. In case both light path and Internet connections are possible the scheduler can try both transport networks and the number of hops for the connecting shortest path is retrieved from the knowledge base too. For reasons of simplicity we take $E_{LAN\_local\_dc}$ equals to $E_{LAN\_remote\_dc}$ and $P_{comp.host\_local\_dc}$ equals to $P_{comp.host\_remote\_dc}$. In an implementation of a scheduler, the scheduler will have knowledge of its own data center and all values concerning a remote data center will be retrieved by issuing a proposal to the scheduler of the remote data center. In that case, values for local and remote equipment maybe different.

We will illustrate a decision made with an example, where the local data center, with $PUE_{local\_dc} = 1.4$, is situated in the Netherlands and is powered by electricity produced from natural gas (380 gr. CO_2/kWh). Suppose the only alternative at the disposal of the scheduler is a remote data center in Tirol, Austria, that is powered by hydroelectricity (15 gr. CO_2/kWh) and $PUE_{remote\_dc} = 1.8$. Values for the connecting transport network can be prepared as knowledge to the scheduler in the following way. If the transport connection between the Netherlands and Tirol has 4 hops, then $E_{network}$ = 0.0014 kWh/GByte for an Internet connection and $E_{network}$ = 0.00066 kWh/GByte for a light path connection. For

$PUE_{network}$ we use a default value of 2.2 (a value based on a recent survey [74], where we assume that more effort is put in data center equipment than in scattered network equipment), while for $X_{network}$ we use an estimate based on the shortest geographical paths between the countries and the information on the typical energy sources used in the countries crossed. In our example, the shortest path long distance network will most probably traverse the following three countries: the Netherlands, Germany and Austria. From data published by the European Commission [81, 82, 83] the energy production in the Netherlands, Germany and Austria is composed by the mixes depicted in Figure 29.

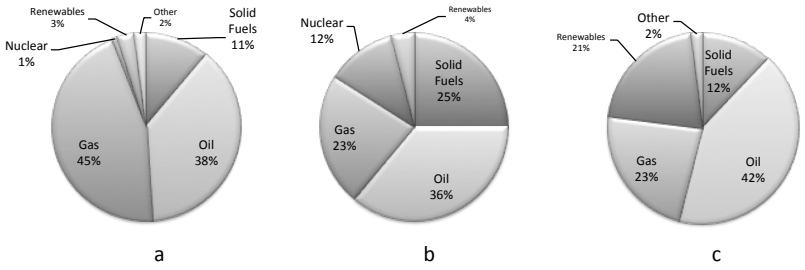


Figure 29: Energy production mix for (a) the Netherlands, (b) Germany and (c) Austria.

From these mixes we derive a mean value for the emission cost in gr. CO_2/kWh. For instance Germany use 36% Crude Oil (640 gr. CO_2/kWh), 25% Solid fuels (pulverized coal 870 gr. CO_2 /kWh), 23% gas (380 gr. CO_2/kWh), 12% nuclear (66 gr. CO_2/kWh) and 4% renewable (30 gr. CO_2/kWh), arriving at a mean value $X_{network}$ Germany = 549 gr. CO_2/kWh. In the same way $X_{network}$ the Netherlands = 520 gr. CO_2/kWh and $X_{network}$ Austria = 474 gr. CO_2/kWh. The distance from say Amsterdam to Tirol is 980 km, of which 120 km in the Netherlands, 600 km in Germany, and about 260 km in Austria, or 12%, 62% and 26% respectively. So, these numbers give an estimate for the transport network $X_{network} = 0.12 \cdot 520 + 0.62 \cdot 549 + 0.26 \cdot 474 = 526$ gr. CO_2/kWh. Imagine a user submits a task needing a lot of experimental data, say $N_{in} = 10$ GByte, and producing $N_{out} = 2$ GByte of graphical data during 0.12 CPU core hours. The scheduler will respond to the user with detailed information it based its decision upon. Figure 30 shows the output the scheduler provided to the user.

In Figure 30 we see also values associated to the energy production in the country of the data center. Models used are not discussed in this chapter, but can be retrieved from a report [76]. The contribution of the LAN of the local data center and of the network, occurring on the right hand side of equation 15, due to the transport of the input data, turn out to be a considerable part of the total energy consumption. This contribution will be even higher if an Internet connection

Scenario 'software (interactive)'
Lightpath long distance

PUE=1.4
Natural gas
X: 380 gr. CO_2 per kWh

Cost local data center	
	g CO_2 (kWh)
LAN (input data)	11.18 (0.0294)
CPU	22.66 (0.0596)
LAN (output data)	2.24 (0.0059)
Energy prod. loss	1.46 (0.0038)
Total	37.54 (0.0988)

PUE=2.2
X: 526 gr. CO_2 per kWh

PUE=1.8
Hydroelectricity
X: 15 gr. CO_2 per kWh

Cost to remote data center	
	g CO_2 (kWh)
LAN local (input data)	22.37 (0.0589)
Network (input data)	7.66 (0.0146)
LAN remote (input data)	0.57 (0.0378)
CPU remote	1.15 (0.0767)
LAN remote (output data)	0.11 (0.0076)
Network (output data)	1.53 (0.0029)
Energy prod. loss	0.07 (0.0049)
Total	33.47 (0.2034)

Figure 30: Detailed output from the decision of a scheduler, the left and right table correspond respectively to the left-hand and right-hand side of equation 15. Remote processing of the job has a lower carbon footprint if the connecting network is a light path network

was chosen, that due to the relative high power consumption of the routers in the network path. If the user knows how the computation and its output data scale with the amount of input data, equation 15 can be applied on a range of input data to see how the cost of the different components scale.

5.5.1.1 *Data ranges and complexities*

We introduce the complexity of a task where both the computation time and the output data scale with the input data, and define $T_{processing} = f(x)$ and $N_{out} = g(x)$ with $x = N_{in}$. For a task with processing time and output data both scaling linearly with the input data, $O(x)$, we have $f(x) = f_1 \cdot x + f_0$ and $g(x) = g_1 \cdot x + g_0$. For a task exhibiting a processing time scaling quadratically, $O(x^2)$, and output scaling linearly, $O(x)$, we have $f(x) = f_2 \cdot x^2 + f_1 \cdot x + f_0$ and $g(x) = g_1 \cdot x + g_0$. In case the amount of input data x is specified or expressed as a range, *i.e.*, $x \in [X_0, X_1], X_0 > 0$, and the complexity of the job is specified, *i.e.*, $f(x)$ and $g(x)$ are specified, equation 15 will decide whether local or remote processing is preferable for each $x \in [X_0, X_1]$. With these definitions we can facilitate a user or the operators of a data center in their choices of task placement with more flexible parameters. The framework has a web calculator which allows data ranges as input for the amount of input data of a task and complexity formulas for the CPU processing time and the amount of output data as a function of the input data.

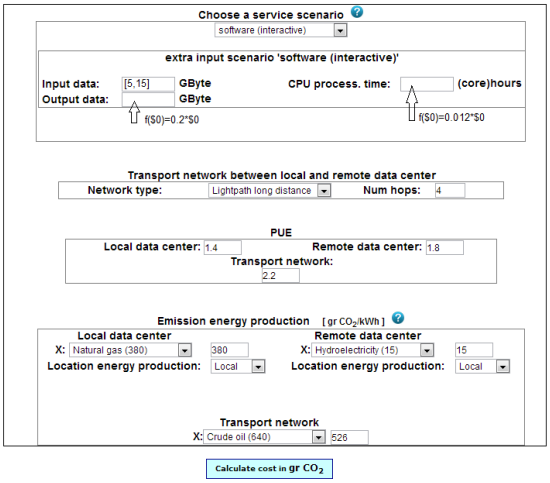


Figure 31: Web calculator for a user or operator to decide whether a task can be greener performed at a remote data center instead of at his local data center. Input data is defined as a range, output data and CPU processing time are defined as complexity formulas on the input data range (the symbol $o refers to a value in the input range).

5.5.2 *Web calculator*

The web calculator[1], facilitates a user to study the output from the scheduler on submitting a task, and also to survey for which amount of input data decisions may alter. As an independent tool the user should supply all the data. Operators of a data center may use data from a knowledge base. We will introduce the web calculator according to the example used so far. Figure 31 shows the web calculator input page. The amount of input data is expressed as a range, [5,15] GByte, and the CPU processing time exhibits a linear complexity, $O(x)$, on the amount of input data, $0.012 \cdot x$, where x refers to a value in the input range. The output data also shows a linear complexity, $0.2 \cdot x$. So we assume that computation time and amount of output data is negligible small if no input data is present ($f_0 = g_0 = 0.$). For $x = 10$ GByte we have CPU time equals $0.012 \cdot 10 = 0.12$ core hours and output data equals $0.2 \cdot 10 = 2$ GByte, values used above. In case a range is defined as input the calculator responds with a plot, Figure 32, and table output for the largest value of the range, see Figure 33.

An operator might use the web calculator to study what happens if the light path long distance transport connection is not available and an Internet long distance connection is the only option. If he keeps all input the same except for the connecting transport network, and choose Internet long distance instead of light path long distance, he notices from the output, Figure 34 and 35, that the decision changes.

1 See http://sne.science.uva.nl/bits2energy/index.html

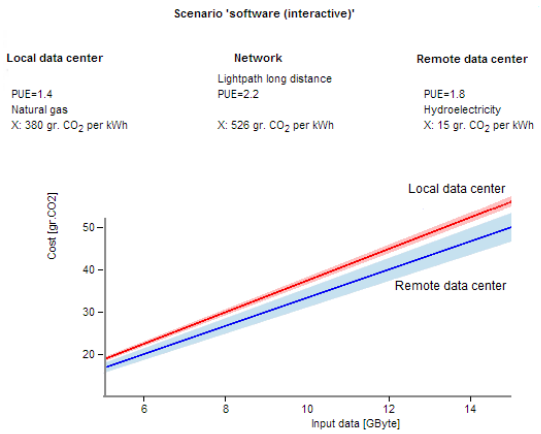


Figure 32: Graphical output of the web calculator if the input (Figure 31) has a range define on the input data. The shaded area is due to an adopted error in the carbon emission value per kWh.

Scenario 'software (interactive)'

Lightpath long distance

PUE=1.4
Natural gas
X: 380 gr. CO_2 per kWh

PUE=2.2
X: 526 gr. CO_2 per kWh

PUE=1.8
Hydroelectricity
X: 15 gr. CO_2 per kWh

Cost local data center	
	g CO_2 (kWh)
LAN (input data)	16.78 (0.0441)
CPU	33.99 (0.0895)
LAN (output data)	3.36 (0.0088)
Energy prod. loss	2.19 (0.0058)
Total	56.31 (0.1482)

Cost to remote data center	
	g CO_2 (kWh)
LAN local (input data)	33.55 (0.0883)
Network (input data)	11.49 (0.0219)
LAN remote (input data)	0.85 (0.0568)
CPU remote	1.73 (0.1150)
LAN remote (output data)	0.17 (0.0114)
Network (output data)	2.30 (0.0044)
Energy prod. loss	0.11 (0.0074)
Total	50.20 (0.3050)

Figure 33: Values corresponding with the maximum value of the input range [5,15] GByte for web calculator input of Figure 31.

The Internet long distance transport network spoils the greener processing advantage of the remote data center.

For quadratic behaviour of the computation time it turns out that it becomes profitable to do the computation at a cleaner remote data center for even modest complexity values. This is due to the fact that the power consumption of computation nodes is relatively high. We saw that there is a difference if one compares Internet with dedicated light path connections due to the power consumption of routers in the former. This becomes clear if we transform equation 15 into a

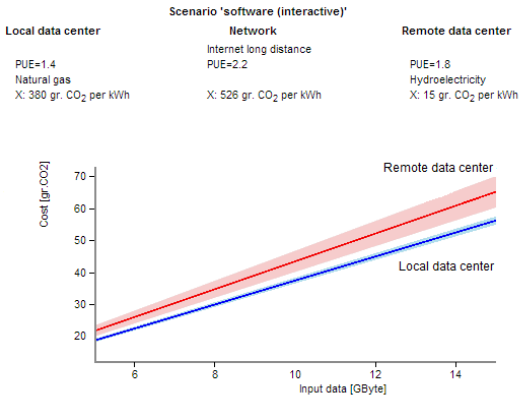


Figure 34: Graphical output of the web calculator if the input (Figure 31) has a range define on the input data, and the connecting transport network is an Internet long distance network (4 hops).

Scenario 'software (interactive)'

Local data center	Network	Remote data center
PUE=1.4 Natural gas X: 380 gr. CO_2 per kWh	Internet long distance PUE=2.2 X: 526 gr. CO_2 per kWh	PUE=1.8 Hydroelectricity X: 15 gr. CO_2 per kWh

Cost local data center	
	g CO_2 (kWh)
LAN (input data)	16.78 (0.0441)
CPU	33.99 (0.0895)
LAN (output data)	3.36 (0.0088)
Energy prod. loss	2.19 (0.0058)
Total	56.31 (0.1482)

Cost to remote data center	
	g CO_2 (kWh)
LAN local (input data)	33.55 (0.0883)
Network (input data)	24.07 (0.0458)
LAN remote (input data)	0.85 (0.0568)
CPU remote	1.73 (0.1150)
LAN remote (output data)	0.17 (0.0114)
Network (output data)	4.81 (0.0092)
Energy prod. loss	0.11 (0.0074)
Total	65.29 (0.3337)

Figure 35: Values corresponding with the maximum value of the input range [5,15] GByte for web calculator input(Figure 31), and the connecting transport network is an Internet long distance network (4 hops).

decision boundary, i.e. substituting an equal sign for the greater sign in the formula.

If we assume linear complexity for input and computation time, where we took $N_{out} = g_1 \cdot x$ and CPU processing time is $f_1 \cdot x$, with $x = N_{in}$, the decision boundary becomes a function of g_1 and f_1, because x cancels out. The result is then visible in Figure 36, with two decision boundaries, $f_1 = 1.43 \cdot 10^{-2} + 4.24 \cdot 10^{-3} g_1$ for Internet and $f_1 = 9.56 \cdot 10^{-3} - 5.28 \cdot 10^{-4} g_1$ for light path. We see three regions corresponding to different choices of task location. In region 1 the task

should be performed locally, independently of the type of transport network; in region 2 the task can be performed remotely provided that the connection is a light path; in region 3 the task should be done remotely for both types of transport networks. Values of the example chosen above, $f_1 = 0.012$ and $g_1 = 0.2$ give a point in region 2, a different decision for light path and Internet long distance transport network.

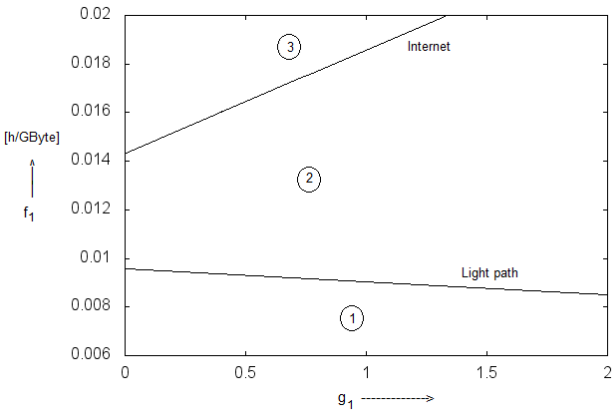


Figure 36: Decision boundaries according to equation 15 for Internet and light path connections with 4 hops.

5.6 DISCUSSION

As we had foreseen in the Introduction the PUE of two data centers, and even their power sources, cannot be the only guiding criteria in choosing the location of a computation or of data storage task. In case the transport network between them is powered by dirtier energy than both data centers are powered with, the contribution of the network to the total cost in gr. CO_2 for moving data can be significant. This mostly is the case if the data traverses the Internet, due to the relatively high power consumption of routers. Light path connections are preferable over Internet connections, but light path connections are dedicated connections that require a more complex setup procedure and sometimes might not be available to a user. For large input data sets and linear behaviour of the computation time on the input data, it might be better to do the calculation locally, if the connecting network is Internet. The same situation may be reversed in case the computation time shows a quadratic dependency on the input data. In that case the contribution of a dirty network becomes less prominent provided the data produced by the computation is limited and does not need to be transferred back to the user. Altogether this means that for realistic large processing, there is not one choice

that can be made that is "always best" in terms of energy use and associated emissions.

5.7 CONCLUSIONS AND FUTURE WORK

We have presented in this article a decision framework to allow users and data center operators to decide where to place an application in order to minimize the total CO_2 emitted in the process. We have shown that, if one assumes that the two data centers being considered have the same architecture and internal structure but different PUE, the network connection between them can play a significant role for the final selection of the site in which to compute or store data. Our framework depends not only on the models for the networks, which can be enhanced if one wishes, but also depends on the contents of the knowledge base it can draw upon. In the work presented here we used the energy data published by the EU and data of some European continental data centers. There are improvements we intend to include in our framework in order to obtain even more realistic carbon footprint information. For data centers that are only reachable by crossing seas, the network model should be enhanced by models of sea cables. Another aspect connected with the network topology used in the models is the knowledge of the exact numbers of hops between two locations. For this, we would like to use a detailed map of the networks for different countries. Our first step in this direction will be to fill the knowledge base with detailed information of the transport topologies used between higher education and research data centers in the Netherlands, which are connected by the SURF network[91].

6

MAXIMIZING THE USE OF ALLOCATED RESOURCES

This Chapter is based on "Software Controlled Virtual Infrastructure"[92]. A software defined internet comprises an IP-tunnel network, software routers on virtual machines (VMs) and Netapps. The previous chapters described concepts that allow to construct Netapps that yield virtual internets that feature localisation and security services on basis of cPUFs as well as optimal distribution, networking and ecological cost of VMs deployed. This Chapter shows how to use allocated VMs most effectively which benefits their performance as well as their energy efficiency. In previous chapters Netapps controlled virtual internets. In this chapter we expand the concept to control a virtual computing infrastructure: the virtual internet as well as the applications that execute on the VMs. For the clarity of the publication contained in this chapter, the Netapp is named there "VI-controller". Amongst others, the chapter shows that virtual internets are great execution environments for workflows. An application specific and a general applicable algorithm in the VI-controller jointly optimize the processing capacity of the workflow. The Warehouse process is application specific as it addresses, with a specific algorithm, the functional concern to determine the cost, the number of VMs and their distribution over cloud data centers. Developers of Warehouse processes can use the previous chapter to determine the environmental costs of a set of VMs in various data centers. The use of the methods in earlier chapters would allow to determine an optimal virtual internet topology. Separated from the design of the Warehouse process is the common concern to use its allocated VMs optimally, that is, to operate the workflow at maximum throughput. For this we present a generic algorithm. This chapter concludes the research in this thesis that establishes that virtual internets are practical execution environments for global scale distributed applications.

6.1 INTRODUCTION

Being available globally, public cloud data centers provide the processing and distribution capacity for virtual machines (VMs) that host applications that interwork to form a distributed workflow. On that global scale, the processing performance of VMs [93] alter, as does the size and nature of the gathered and produced data streams. For example, daylight conditions influence the compression of video streams and the output of recognized features. Hence, to maintain a most effective use of the available VMs, the application must be scaled and distributed regularly.

The horizontal scaling mechanism is often available at cloud data centers. This mechanism replicates VMs, and therefore the applications they execute, in a certain data center. The horizontal scaling mechanism is frequently used to adapt the number of webservers to provide web clients enough quality of service. More complex distributed applications, such as the data analysis framework Spark[94] and the streaming-data analysis framework Storm[95] are implemented as workflow applications. They use the resource manager Yarn[96] to scale and distribute, at instantiation time, the applications that constitute the workflow. Yarn ignores run-time application performance indicators that are affected by a change in the nature of data (e.g., larger and more complex data packets) and VMs that receive less CPU time or memory bandwidth. In this way Yarn omits the complexities that run-time scaling of applications introduce, namely that of gracefully integrating or removing the just amount of applications in and from the workflow. The designers of a globally distributed and adaptive content replication network, described in ref.[97], benefitted from the deployment of a virtual network. Their virtual network overlays telecommunication networks that interconnects their private cloud infrastructure. The deployment of content replication applications is straightforward in this virtual network. For example, the virtual network hides the changes in telecommunication links caused by the activities of software to obtain optimal network latency, capacity and robustness.

This chapter shows how the construction of a globally distributed workflow benefits from a Software Controlled Virtual Infrastructure, SCVI. The SCVI comprises VMs in public cloud data centers and IP-tunnels that interconnect them. As in the case of the adaptive content replication network, a Virtual Infrastructure (VI) controller can construct IP-tunnels that have the same or better performance than default internet connection [23] and manage their robustness[2]. A workflow specific Warehouse process in the controller determines, run time, the number of VMs and their distribution over cloud data centers. As a response to fluctuating workload and VM performances, the controller reconfigures the workflow and the IP-tunnel network to ensure optimal usage of the allocated VMs.

We discuss the following software:

1. parts of the VI controller that warehouses VMs and deploys them most effectively in a workflow via a generic optimization algorithm,

2. Sarastro[2] software the VI controller uses to allocate VMs and to install, on them, router software, IP-tunnels and applications,

3. Pumpkin[98] a distributed, peer-to-peer workflow manager, steered by the VI controller, and

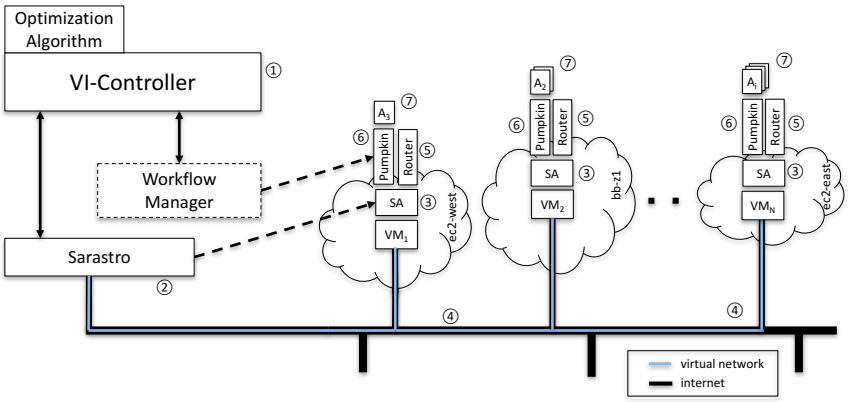


Figure 37: The VI-controller (1) controls the life cycle and optimization of the workflow. Its Warehouse process determines the number of VMs and the set of data centers from which they are acquired. The optimization algorithm of the VI-controller ensures effective use of them. The VI-controller uses Sarastro (2) to allocate and bootstrap VMs. VMs run Sarastro Agent (3) allowing to setup and manage an Internet overlay network (4) and router software (5) to interconnect the VMs. Similarly, each VM contains an instance of Pumpkin, a peer-to-peer workflow manager (6), to allow the VI-controller to add, run and remove different applications A_i (7). The Pumpkin instances together form a virtual workflow manager that constructs and optimizes a workflow between the A_i.

4. VMs that are configured to host workflow applications and that are managed by the VI controller.

6.2 ARCHITECTURE

Our technology is relevant for the implementation of workflows that feature horizontal scaling. This requires the externalisation of the workflow state from the scaled applications. Externalisation of state prevents the destruction of state information when applications are removed from the workflow during a scale-down operation. In our architecture, see Figure 37, state is maintained in the VI-controller and in the workflow packets. Then the act of scaling and distribution is essentially the replication and distribution of VMs and the attachment and detachment of applications from the workflow. To generate the proper network connections, the VI-controller contains the logical topology, e.g., Figure 38a, that describes how the applications in the workflow connect to each other. This allows the VI-controller to create, via Sarastro, a network of IP-tunnels that connects a set of VMs. The tunnels and VMs form a virtual infrastructure that overlays Internet and computer hardware. This virtual infrastructure forms the execution environment of the workflow. The management of details

of the underlying network and computing environment is left to the VI-controller and Sarastro. The applications in the workflow merely experience a routed IP-network, to which instances of applications in the workflow connect and disconnect. On each VM there is a Pumpkin instance that follows instructions of the VI-controller to manage a single active instance of an application in the workflow. Figure 37 reflects this: A_{ij}, the j-th instance of a workflow application A_i is controlled by its Pumpkin instance. As Pumpkin instances abstract A_i to the VI-controller, the performance of the workflow is optimized on basis of an abstract performance criterion. This criterion, the time to process all workflow packets in the input queue of A_{ij}, is determined by its managing Pumpkin instance and reported by this instance to the VI-controller.

6.2.1 *Sarastro virtual infrastructure factory*

On behalf of instructions from the VI-controller, Sarastro[99] instantiates VMs on a designated cloud data center. All VMs are based on an image on which the Linux OS and Sarastro Agent are configured. During run-time, the VI-controller uses Sarastro to manipulate the network of IP-tunnels as well as to manage VMs. Furthermore, on behalf of the VI-controller, Sarastro terminates VMs.

6.2.2 *VMs*

Sarastro manages software on VMs via the Sarastro Agent that is installed as an application. Router, Pumpkin and A_i instances are placed via this agent on the VM. Furthermore, the VI-controller configures, via the agent, IP-tunnels to pre- and succeeding applications in the workflow on basis of its logical topology, see for example Figure 38a. In cases the Pumpkin instance is commanded to replace an instance of A_i with one of A_j, the tunnels are reconfigured following the prescription of the workflows logical topology too. Each IP-tunnel is assigned an IP address and supports IP-broadcasts on the subnet. As the tunnel interfaces are also registered with the router on the VM, the IP-tunnels and the routers form a routable subnet, where every VM is reachable via two or more paths.

6.2.3 *Pumpkin workflow manager*

Pumpkin[98] [100] is a decentral, peer-to-peer workflow manager. Pumpkin instances broadcast over the IP-subnet to discover the addition or removal of preceding and succeeding applications in the workflow. Following the concept of a distributed data transformation network (DTN), Pumpkin tags a data packet with a state automaton and marks its current state to create a workflow packet. A Pumpkin instance that

receives a workflow packet, instructs its A_{ij} to process it and updates the automaton. Then the automaton is added to the output data of A_{ij} and forwarded to an appropriate succeeding application. This continues until the automaton reaches its termination condition, i.e., until the workflow is completed for that data packet. Pumpkin instances implement, per succeeding application, a limiter on the rate packets are sent. The rate limiter measures and stabilizes the PE_{ij} of the succeeding application, the estimated time this application processes the workflow packets that are currently stored in its input queue. The combined effect of the rate limiters is a continuous minimization of all backlogs. Each Pumpkin instance is also queried for PE_{ij} by the VI-controller that harbours an algorithm to optimize deployment of allocated VMs. This algorithm determines which A_i to run on each VM. Pumpkin facilitates the distribution and scaling of A_i. To scale up A_i with an additional instance, the VI-controller merely has to instruct a Pumpkin instance on an unused VM to load A_i. In the case that, e.g., for cost reasons, the number of VMs has been fixed and all of them are in use, the VI-controller instructs a Pumpkin instance on a certain VM to unload its active A_j instance and load one of A_i. In this case, the increase of capacity of A_i is at the cost of A_j's capacity.

6.2.4 *VI-controller*

The Python coded virtual infrastructure (VI) controller [101] creates, monitors and manipulates virtual infrastructures as well as the workflow. The services of Sarastro are used to manage the virtual infrastructure. Furthermore, the VI-controller interfaces with each Pumpkin instance to manipulate the workflow. The VI-controller contains a purpose specific implementation of a Warehouse process that determines, at what cost, how many VMs should be acquired at which data centers. Cost, both financially as environmentally [72], is a reason to constrain overdimensioning of the workflow. Furthermore, cloud data centres also limit the number of VMs deployed, e.g., the default maximum of Amazon EC2 was 20 VMs at the time of our experiments. During the run time of the workflow, the numbers and distribution of VMs can be varied by the Warehouse process. The VMs are started by the VI-controller, causing amongst others, activation of their Sarastro Agent and Pumpkin instance. The controller instructs each Sarastro Agent to connect the VM with IP-tunnels to a VM that will contain the workflows data-source A_1, see Figure38b. Having established a network in this way, the VI-controller builds, guided by the logical topology and starting with the data source application A_1, the workflow, see Figure38c. Each building step includes the activation, via Pumpkin, of the appropriate A_i instance A_{i1} on a VM and the creation of an IP-tunnel to the preceding application A_{j1}. The IP-tunnel between each A_i instance to the ones of the data-source A_1, is re-

moved in case the logical topology the A_i does not have a link to A_1. After the establishment of an IP-tunnel, Pumpkin instances discover each other as well as the A_{ij} that they manage. Once the initial workflow system has been set up, all A_1 instances are instructed to start the injection of data. About 20 seconds after the workflow system starts executing, all predicted execution times PE_i are stable. Then the VI-controller deploys a generic optimization algorithm to ensure that the allocated VMs process data most effectively. This results in a scaled workflow system, see 38d-e.

6.2.5 *Generic optimization algorithm*

The optimization algorithm operates on non-application-specific information to achieve a most effective use of the VMs allocated by the Warehouse process. It requires the identifier ij of the instances A_{ij} and their predicted execution time PE_{ij}. Under governance of the optimization algorithm, the VI controller changes the number (n_i) of A_i instances in such a way that the average predicted execution times $PE_i = \sum_j PE_{ij}/n_i$, for all A_i, are approximately equal. Then no A_i forms a performance bottleneck, the rate in which packets are entering the workflow is equal to the rate of processed packets, the workflow is in free flow.

Basically, the algorithm that determines the values of n_i for the free flow state, works as follows. It first tries to assign an unused VM to run another instance of the A_i with the lowest PE_i. However, if all VMs already run an A_i, the VI-controller commands an arbitrary VM from the set of VMs that run an instance of the best performing A_i to terminate it and run an instance of the worst performing one. The controller repeats this until the free flow state is achieved. More precisely, the VI-controller

1. collects the predicted execution times PE_{ij} from the Pumpkin instances.
2. calculates the averages PE_i.
3. sets i_{max} to the index i of the maximum PE_i. $A_{i_{max}}$ is the application that currently forms the bottleneck in the workflow.
4. sets i_{min} to the index i of the minimum of PE_i for A_i where $n_i > 1$.
5. configures, in case there is an unused VM, this VM to run an instance of $A_{i_{max}}$.
6. configures, in case if every VM is used, an arbitrary VM that runs an instance from the set $A_{i_{min}}$, to run $A_{i_{max}}$.
7. waits for a certain amount of time and then continues at the first step.

The VI-controller steers the workflow's processing capacity towards the optimum state, a state where the input queues of each A_i are processed in the same amount of time. In this workflow state the available VMs are exploited to their maximum effectiveness. If nevertheless the queues are growing in this state, the processing capacity is insufficient, and vice versa. As the Warehouse process selects the datacenters at which the VMs are deployed, it determines the geographical distribution of A_i. The optimization algorithm ensures that the distributed A_i are balanced in performance globally.

6.2.6 *Simulator*

Much of the behavior of the optimization algorithm can be studied via a simulator[3]. For instance, Figure 38 illustrates the initialization phase, several stages in the scaling phase and the final equilibrium state of a simulated workflow system.

By observing various simulated workflows, we recognized that the topology of a workflow at equilibrium profiles the characteristics of the data and qualities of the virtual infrastructure. In fact, the actual workflow topology is defined by the logical workflow topology and the performance profile, the set of $\{n_1, n_2, ...\}$. Here, n_i is the number of instances of A_i. The graph in Figure38e has $\{1, 4, 2, 2, 4, 3, 4\}$ as its performance profile. The performance profile depends on processing speed of the A_{ij}, and the nature of the data streams they operate on. If, for instance, the coding of the workflow application A_i improves its processing capacity then less VMs are required to process its workload. The VI-controller would attribute the excess VMs to other workflow applications. Hence, the performance profile is changed. Similarly, if the nature of data changes it affects the processing speed of the workflow applications A_i and that is reflected in the performance profile too. The performance profile is a characteristic that fingerprints the workflow process as a whole. Changes in the performance profile can be used to trigger an action. The workflow simulator is useful also for other developers that deploy our software. It allows, amongst others to evaluate specific remedies, e.g., for oscillations or fluctuations in a workflow topology that are caused by the discrete nature of the optimization algorithm.

6.3 EVALUATION

A SCVI provides an adaptive execution environment that can be used to right-scale and distribute, globally, applications. In the case discussed, the applications form a workflow that is managed by Pumpkin. Easing the efforts to engineer and operate the workflow is an architectural concept that allows separation of concerns in the design phase. During design time, the developers who design the workflow

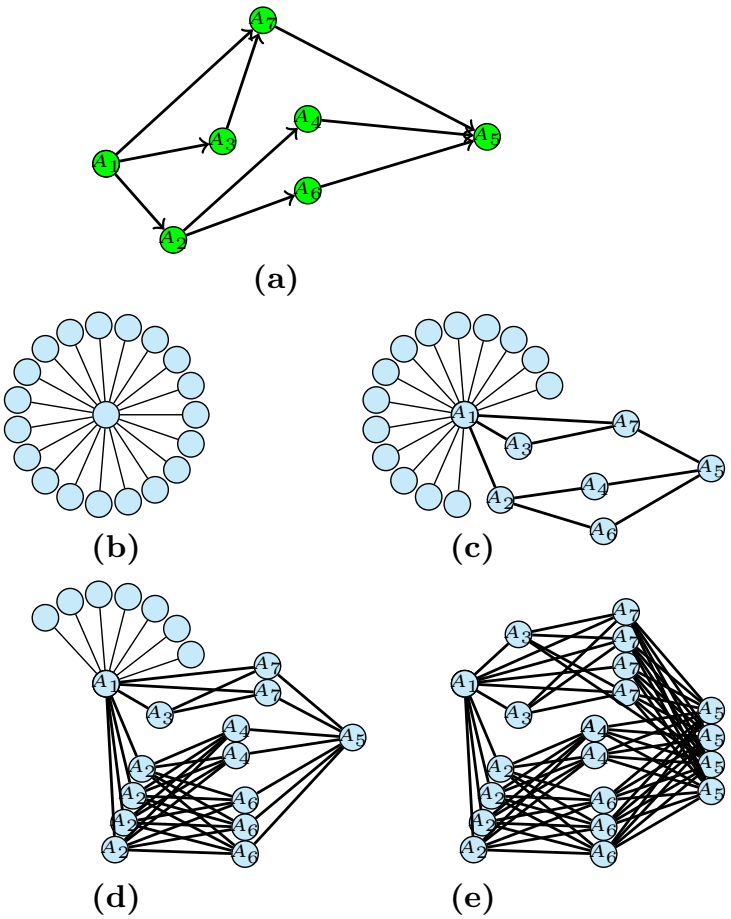


Figure 38: Effects of the VI-controller on the scaling of a simulated virtual infrastructure of a distributed workflow application. (a) Is an acyclic directed graph, which is the logic workflow topology. (b) Is the virtual infrastructure topology of IP tunnels and VMs setup by the VI-controller. Then (c) the controller sets up the initial, non-scaled, virtual internet topology running an active workflow. This topology partly reflects the logical workflow topology shown in (a). (d) Intermediate topology after a few scaling actions. (e) Is the final equilibrium topology. Application A_1 is the data injector.

and develop the A_i, can be others than those who adapt the VI controller to generate, scale and distribute the workflow. During run time we have taken care, as we will discuss, that the scaling algorithm of the VI-controller and Pumpkins adaptive mechanisms do not disturb each other. Amongst others, Pumpkin features feed-back mechanisms to adjust rate limiters, to cope with dropped workflow packets and to deal with the effects of applications entering and leaving the

workflow. To evaluate the practical applicability of the architecture and possible side effects of interworking adaptive systems, a demonstrating application is created. The existing and hence independently developed Twitter filter engine that had served as a demonstrating application[98] of Pumpkin itself is used for the purpose.

6.3.1 *Twitter filter workflow*

The Twitter filter workflow is designed to process data from a 25GB Twitter data set [102]. The workflow is composed of the following A_i, of which A_2, A_3, ..., A_5 act as filters:

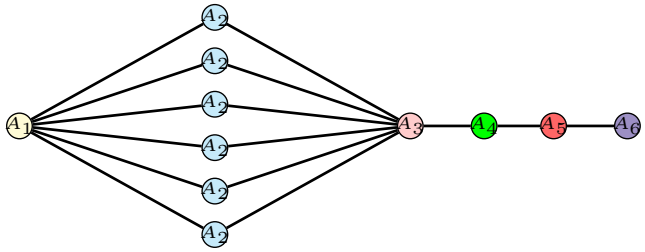
- A_1 Injector - of data in the workflow.
- A_2 Language filter - e.g., English.
- A_3 Sentiment filter - to categorize tweets into positive and negative sentiments.
- A_4 Topic detection - to match a tweet to a predefined set of topics.
- A_5 Entity recognition - to extract named entities from tweets such as movie titles, brands, etc.
- A_6 Data collector.

The software engineering of the VI-controller of the Twitter filter workflow profited indeed from the separations of concern. The logical workflow topology that is coded in the VI-controller, is trivial: A_1 outputs to A_2, A_2 to A_3 and so on. There is much performance variability in the application, and this tests the adaptive mechanisms. The A_i, for instance, differ in computing resources to process a workflow data packet and foremost A_2 is computational intensive. Also the CPU times assigned by data centers to VMs fluctuate. Then, as the Twitter data themselves differs in composition, their processing times differ too.

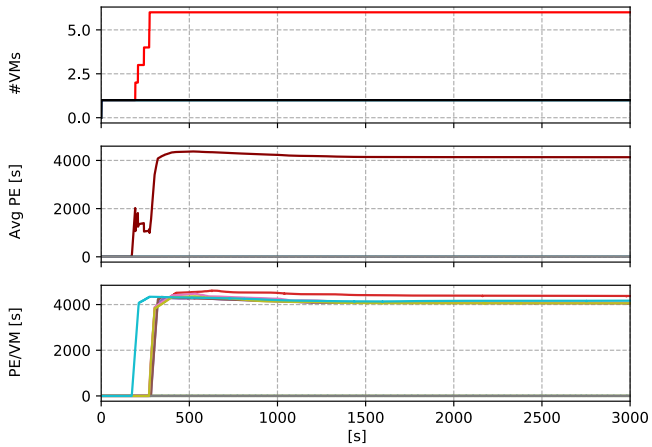
6.3.2 *Experiments*

Various experiments have been conducted. The most extensive one allocated VMs from two cloud providers, namely Amazon EC2 and Brightbox. The Warehouse process obtained five VMs each from the following data centers: Amazons US West, US East and EU West and Brightbox's Zone A. Each VM has 1 virtual CPU and 2 GB of memory. The VI-controller follows the procedures outlined in Section 6.2 to setup and scale the workflow.

In this experiment A_2, the English Language filter, got finally 10 VM assigned to it. The other five VMs were assigned by the VI-controller to the other five A_is. This indicates that A_2 is by far the



(a)



(b)

Figure 39: Figure 39a shows the equilibrium workflow topology with the following applications, A_1 (Injector), A_2 (Language filter), A_3 (Sentiment filter), A_4 (Topic detection), A_5 (Entity recognition), and A_6 (Data collector). Of the eleven available VMs, the optimizer allocates at least one VM to each A_i and attributes the remaining ones in such a way to the A_i that the throughput of the workflow is optimized. Here, these VMs run A_2 as this application is by far the most compute intensive application. Figure 39b shows the average predicted execution times, PE_i, of all workflow applications $A_1, \ldots, A_6$. The execution times of A_2 instances dominate the graph, the ones of the other applications are barely visible.

most compute sensitive application. Furthermore, the experiment revealed a cross-cutting concern. As the input buffer size of A_2 was set too large, the buffers of A_2 instances got populated unevenly. To understand this, suppose that workflow packets are sent at a rate R to a certain A_i instance, that the A_i instances have an infinite input buffer size and that they are able to process packets at a rate $r, r \ll R$. The input buffer of the first instance of A_i, A_{i1} is filled at an rate $R - r$. After t_c seconds, the cycle time of the scaling mechanism, A_{i2} is added by the VI-controller, then $PE(t_c)_{i1} = (R-r)t_c/r = (\frac{R}{r} - 1)t_c$. At $2t_c$ it

is decided if A_{i3} is added on basis of an evaluation that involves the average execution time of A_i, $PE(2t_c)_i = (PE(2t_c)_{i1} + PE(2t_c)_{i2})/2$. Now $PE(2t_c)_{i1} = (\frac{R}{r} - 1)2t_c$ and $PE(2t_c)_{i2} = (\frac{R}{r} - 1)t_c$. $PE(2t_c)_{i1}$ and $PE(2t_c)_{i2}$ are unequal and this reflects their scaling history. Similar effects happen for subsequent scaling actions. Hence, in the case at hand, the average PE_i is affected by details of past scaling actions. This makes comparisons of PE_i, for all values of i, hard to interpret since they do not reflect the contemporary state of the workflow. In the case of the Twitter filter workflow, where the processing requirements of A_2 dominates by far those of the other applications, this still yields a sensible workflow topology. Many other workflow applications require PE_i that does not depend on the history of the workflow. To ensure this, the buffers of each A_i instance must be filled or emptied nearly completely within each scaling cycle, then their average PE_i reflects the current state. By observing the workflow, one can adjust the input buffer size of the A_i instances for this purpose. Alternatively, one can adjust t_c. To illustrate the appropriate behavior of the scaling mechanism, we discuss a straightforward setup that deployed eleven VMs in Amazon's EC2 in Dublin.

6.3.3 *Discussion*

After the start of the flow processing, the VI-controller waits for $\approx$ 20 seconds before initiating scaling actions. This allows to minimize startup effects on the determination of the PE_is. Then, the next and subsequent scaling actions take place at intervals of $t_c \approx 20$ seconds.

The result presented in Figure 39 clearly shows that A_2, the English language filter, is by far the most CPU consuming application in the workflow. In this state the optimization algorithm attributes most VMs to A_2, resulting in an easily understandable final workflow topology, see Figure 39a. Figure 39b shows the average predicted execution times PE_{ij} for all A_{ij}. Yet only the instances of A_2 have on this scale a noticeable magnitude. Their input buffers have a size of 200 workflow packets and these remain filled until, in this case manually, A_1 is ordered to stop streaming packets to them. Then the PE_{2j} decrease. The instances of all other workflow applications process their input at such speed that their input buffers remain near to empty, and hence, their $PE_{i1} \approx 0, i = 1, 3, 4, 5, 6$. The PE_{2j}, see Figure 39b, exhibit an overshoot at the beginning of the workflow, which is caused by an interplay between the OSs on the VMs that has to adapt to their new load, and the rate limiters of A_1.

6.4 SUMMARY AND CONCLUSIONS

The construction of globally distributed applications is enabled by the large amount of data centers that, via the internet, offer to run VMs.

We show that the development of such applications profits from separation of concerns that are the result of using software controlled virtual infrastructures (SCVI). With workflows managed by the peer-to-pear Pumpkin software we illustrate how the SCVI provides a straightforward to use virtual networking and computing environment with general applicable mechanisms to add, distribute, scale and remove applications that run on VMs. We developed a generic algorithm to deal with the concern to use the available VMs effectively and, consequently, to distribute and scale the workflow over them. An important cross-cutting concern is identified and remedied. Namely, to ensure that the controller scales the workflow on information reflecting its current state, we maintain small input buffers of the workflows applications. The workflow is robust, the links to pre- and successor nodes always guarantee a backup connection between workflow nodes, whilst running multiple instances of an application provides robustness against application failures.

7 CONCLUSIONS AND FUTURE WORK

7.1 VIRTUAL INTERNETS

Our research group worked on the concepts of virtual internets since 2005. At that time, virtual internets went by the name 'wormhole'. We thought that wormholes were a curious side result of reasoning developed in the research of programmable networks[1]. An Internet packet could be sent into it, to resurface somewhere else on the Internet. The designation 'wormhole' seemed appropriate. Now we understand that virtual internets are effective constructs in the creation of distributed applications. They provide an execution environment that allows developers to build complex and secure distributed applications. Virtual internets, as can be inferred from this thesis, do not have such performance penalties that limits their use to a very select group of applications. Instead, as the research conducted in Chapters 3 and 4 show, a virtual internet connection has equal or better latency and transport capacity than the default internet one.

In Strijkers et al. [2] we introduced principles to program the topology and life cycle of virtual internets. Virtual routers were generated in the cloud and connected via encrypted IP-tunnels. This formed a network that routed IPv4 and IPv6. Such a construct is termed in this thesis as 'virtual internet'. Strijkers et al. demonstrated that a single person can create a Netapp to set up a globe spanning private network, with hundreds of routers, featuring self-repair. We subsequently demonstrated virtual internets with various other features at several SuperComputing conferences. At these occasions, the audience was amazed: globe spanning, hundreds of routers, self-repairing, looping packets, multicast![**P-7, P-8**]. People realized that if networks were software constructs, they could be programmed to do almost anything. There was also the curiosity how the results could be put to practical use. "What do the virtual internets deliver extra, what is the killer app?" the audience was probably thinking. Hence the people asked politely: "What are the applications?" And there was always that question: "What is the performance?" Now, 2017, virtual internet fun, amazement and curiosity has been replaced by insight and knowledge.

This concluding chapter summarizes the answers to the research questions, states how the results contribute to the practical uptake of virtual internets and which novel applications they enable. We discuss in Figure 40 dynamic access to privacy sensitive information, in Figure 41 a distributed data sharing environment, in Figure 42 the

organization of the future Internet and in Figure 43 how future end-systems attach to them.

7.2 ANSWERS TO RESEARCH QUESTIONS

The aim of this thesis is to develop the computer science of the constructions that make virtual internets a practical framework to assemble and operate a distributed application. The research question was formulated in Chapter 1 as:

> *Can we scale, distribute and adapt virtual internets and embed applications in them to achieve a better than best-effort performance of the distributed application?*

Chapter 6 clearly demonstrated that: an auto scaling, self-distributing, self-repairing distributed system, being itself a serious scientific application, that spanned four data centers in two continents.

The first subsidiary research question, see Chapter 2 is the following:

> *1. How can we entangle virtual and physical machines, and how can we use this entanglement for secure communication purposes?*

For this we developed cPUFs, the security primitives based on the intrinsic, unclonable, properties of physical devices. cPUFs are used to guarantee integrity and confidentiality of computing and networking. They are used by security-sensitive computation and communication processes on systems where all the privileged software (kernel, hypervisor, etc.) is potentially malicious. With other words, on basis of cPUFs services one can be sure if, where, and what software was executed and that secure links are established to the right computers. cPUFs in combination with virtual internets, are pivotal to secure data transactions on data.

Chapter 3 answers the following question:

> *2. Does the optimization of a virtual internet topology result in a better end-to-end performance compared to the best effort path over the Internet?*

With a certain selection of data centers one determines how a virtual internet overlays the internet and what its geographical topology is. The data centers, by means of the VMs that run virtual routers of the virtual internet, define the geographical location of these virtual routers. Chapter 3 describes a Netapp that select datacenters, and in them VMs, that act as network nodes and create with them the best performing virtual internet between end-points. Frequently, the virtual internet even outperformed, with respect to latency, the Internet

path between these end-points. Virtual internets can be more robust than the Internet itself. Internet robustness itself is implemented by services on routers. However, it is not an exception that the owners of the various subnets switch these off. Moreover, the resilience services are not always configured properly to allow interworking across domains. Yet, in the uniform environment of the virtual internet it is much easier to configure the resilience services correctly – it is just an option that needs to be switched on. In more demanding cases one can develop, as we demonstrated in [2], a Netapp that changes the network topology such that there are no articulation vertices and bridges.

For those situations where obtaining a global VPN via a Telco is not an option, one might consider to setup virtual internets via selected data centers as an alternative. Chapter 3 shows that virtual internets are in more than half of the cases faster than the default Internet path. Performance of virtual internets is important for their uptake. Hence, one must answer another performance question: "Is the performance of software routers comparable to that of hardware routers?" Chapter 4 presented a novel computer science concept for table lookups, in general, and answered the question relevant for virtual internets:

> *3. Can we improve the process of routing table lookup such that it does not suffer from the memory wall?*

The answer is yes. The novel concept made the algorithm computational bound, whereas state of the art table look-up algorithms are memory bandwidth bound. Phrased differently, state of the art look-up algorithms suffer from the memory wall. The novel algorithm, compared to the traditional ones, sparsely uses the computer's memory. Instead, the table is entirely mapped on instructions that are loaded in the CPU. This resulted in a performance gain of a factor ~10. We improved the performance by a factor ~100 by implementing the lookup-algorithm on GPUs. Our novel table look-up method is generally applicable, hence the results of Chapter 4 is relevant for software engineering in general and might inspire Computer Science to follow this new approach to speed up other algorithms.

In Chapter 5 we addressed the question:

> *4. How can we quantify the CO_2 footprint of a virtual internet?*

In this chapter we developed a method for estimating CO_2 emission costs for distributed systems. This method can be part of a optimization program that finds the optimum set of resources for a given cost. The environmental footprint of distributed applications is a common concern.

The last research question posed was:

5. How to scale, for a fixed set of VMs, distributed applications to achieve an optimum performance?

In Chapter 6 we developed a SCVI-controller, which scales the applications in the workflow to have a uniform processing time. Instrumental for the scaling mechanism is that the workflow is embedded in a virtual infrastructure composed of IP-subnets and VMs. The effect of operating the SCVI-controller is that the VMs are used most optimally. One can simply improve the performance by adding new VMs, as the controller automatically reconfigures the workflow over all VMs.

7.3 REFLECTION ON THE MAIN RESEARCH QUESTION

Let us revisit the main research question. This question refers to the engineering of a sizable distributed application. From a globally distributed set of datacenters there is a sheer limitless amount of resources available (VMs). Software, implementing a basic adaptive method to instantiate, distribute and scale the application must include and repeat the following steps:

1. Select data centers
2. Obtain the amount of VMs that is a compromise between performance requirements and usage cost, a.o., the environmental footprint calculated via the method of Chapter 5.
3. Use the method of Chapter 3 to create optimal performing virtual internet connections between VMs.
4. Execute the SCVI-controller, described in Chapter 6, to roll out, optimize and redistribute the application.

Developers of secure applications will likely modify these steps, e.g., to include VMs with support for PUFs. They can follow the methods described in Chapter 2 to enable location based services and proof of execution. For virtual internet topologies where high throughput is important, one has the option to run software routers that implement the methods described in Chapter 4, or even select VMs that offer access to GPUs. Given the above summary of this thesis achievements this motivates the "yes" on the overall research question:

Can we scale, distribute and adapt virtual internets and embed applications in them to achieve a better than best-effort performance of the distributed application?

There is also a "yes" from a more elevated, a more scientific point of view and therefore we reflect on the issue of separation of concerns

(SoC). SoC simplifies the construction of the distributed applications that the main research question refers to. SoC, caused by the deployment of a distributed application on VMs connected by a virtual internet, permits application developers to concentrate on functional requirements; software that is specific to the application. SoC allow another group of developers, system engineers, to create software, Netapps, that distribute and optimize the virtual infrastructure of VMs, IP-tunnels and the embedded applications. In the case described in Chapter 6, only minor interactions between those groups of engineers is required to create a global, adaptive workflow.

Figure 40: A mock-up of an augmented reality application that interworks with Netapps that control the complex organization of virtual internets that support autonomous driving. The application shows a local virtual internet that connects applications on the car with those on the lampposts. Here, Netapps dynamically govern the connections (green dashed lines) between the active components, e.g., blue bordered car and green lampposts, and passive ones, e.g., the orange lamppost. On one lamppost the Netapp is indicated by the KPN logo and it collaborates with a navigation application, depicted by the TomTom logo. The two applications have separated concerns. This separation of concerns allows the construction of complex software. For instance, the navigation application can instruct the KPN Netapp to extent the virtual internet to the orange and then to the red lamppost. In the same order the KPN Netapp and the navigation application is copied there, whilst deleting the ones passed. Netapps can dynamically manage access to virtual internets and hence to data accessible via these. This is important, as for instance the right to access private data might change due to accidents.

The SCVI-controller in Chapter 6 is an implementation example of the management of a non-functional, execution quality require-

!h

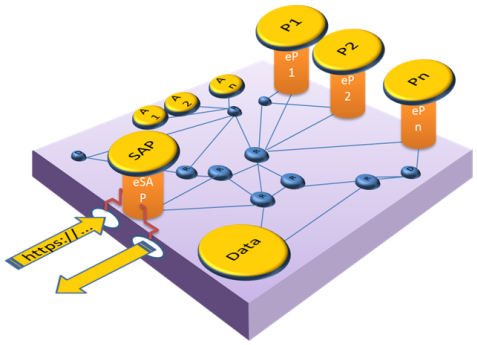


Figure 41: A distributed data sharing environment, a sandbox, on basis of an virtual internet. We envision countless instances of such sandboxes to collaborate and form the foundations of a digital transformed society. Chapter 6 showed that Netapps can generate and run-time modify distributed workflows, including those that run sandboxes. The sandboxes are accessible via a service access point (SAP). In this case the SAP is implemented on basis of a webserver, an enterprise service bus adapter would be appropriate in business environments. Applications $A_1, A_2, \ldots, A_n$ are programs that provide auxiliary services. The environment ensures that data is shared only to programs $P : P_1, P_2, ..., P_n$. Certification of these Ps forms the basis of a mechanism to prevent the use of malicious software. The P_is process data on behalf of several companies, e.g., to bid on the fulfilment of a trade request. Furthermore, the correct, e.g. law-respecting, behavior of the Ps is enforced by additional software eP that runs on the same VM as the programs P. The Pumpkin software described in Chapter 6 is an example of such eP. The sandbox blocks unwanted output of the Ps. Safety is based on the sandbox, its short lifespan, the obscurity of its locations, cPUFs, encryption, and programs that actively monitor security and adapt the virtual internet if needed. The environment uses cPUFs and the methods of Chapter 2 to certify execution of Ps.

ment. This requirement, the optimal use of VMs, establishes a free flow state of the workflow and is achieved by horizontal scaling and distribution of applications in the workflow. The warehouse process described in that chapter implements a functional, evolution quality concern. The warehouse process determines the number of VMs needed at a certain moment. The algorithms to evolve the network are described in Chapter 3), algorithms that calculate the CO_2 cost of execution in Chapter 5.

7.4 OUTLOOK AND FUTURE WORK

Virtual internets that are researched in this thesis make scaling and distribution of applications practical. Virtual internets are generated for each set of collaborating applications. With more and more online devices and appliances, virtual internets will be extended to ever more locations. Virtual internets can be configured as regular Internet subnets. This means that they are either connected to the Internet, to other virtual internets, or both. The application described in Figure 40 uses real and virtual Internets. Virtual internets provide a distributed system technology to implement a most crucial service, "the killer app" that we mentioned earlier: a distributed sandbox, the safe way to share data. Sharing data is basic for most transactions in the digital transformed society. Virtual internets allow control over data access and the set of applications that is allowed to operate on the data, see Figure 41. This concept avoids direct transfer of data between companies, omitting all associated pitfalls. The ability to share data in a safe way is relevant for future machines. These machines are comprised of multiple "smart" parts. Software controls the behavior of these parts, harmonizing their actions. Yet, there are situations where the complexity of these parts demands that their manufacturers control them via the Internet. Hence, the smart part has to be online, whilst the sandbox guarantees that only data from certified applications is exchanged. Here too, the construction shown in Figure 42 serves as a base concept with which ICT of a smart machine is designed.

Our work showed the concepts with which virtual internets provide robust, secure, and well performing execution environments, in fact distributed sandboxes, for distributed applications. Applications that form the essence of a digital transformed society: smart machines, online shopping, government and bank applications. Yet, the transactions these applications perform must comply to law and other regulations, e.g., a digital market should not be a black market. Future work is needed to embed law and rule compliance in these sandboxes, and to re-enforce them.

Finally, Figure 42 and 43 show how an end-user is connected via virtual internets to virtual internets that connect distributed applications. The figures reflect the future architecture of the Internet. When Bill Gates said "The internet is becoming the town square for the global village of tomorrow", we know that virtual internets will fulfil that prophecy. In future, virtual internets are subnets that outnumber their physical counter parts, by far.

A vision of the future internet that includes virtual internet subnets. Most of the Internet subnets reside in the cloud. This thesis shows that virtual internets provides a very flexible, versatile and secure environment to create distributed internet applications.

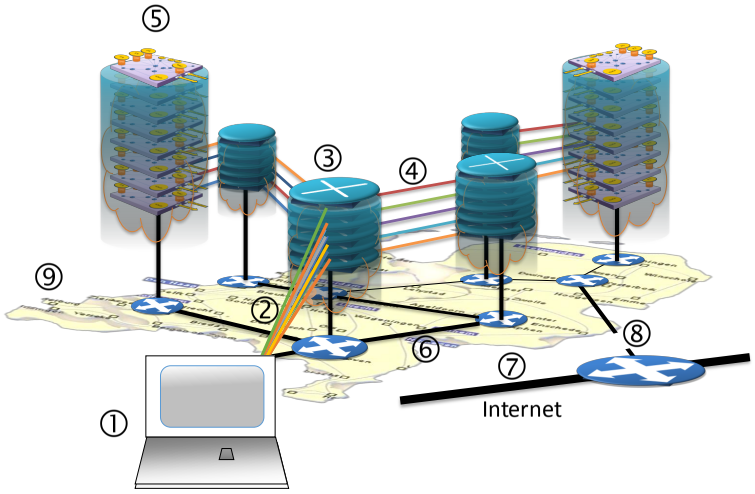


Figure 42

The personal computer ① uses multiple IP-tunnels (colored lines) ② to connected to virtual routers ③. These routers are part of virtual internets ④ in which distributed applications ⑤ are situated. These virtual internets do not necessarily have to follow the same national internet ⑥ topology. For safety reasons some of the virtual internets are not accessible from the internet ⑦. When a connected vitual internet is attacked, one might disconnect it from the rest of the internet ⑧. To increase security, one isolates traffic and provide a straightforward network structure that eases the forensics of incidents. Therefore, virtual internets are mapped on OSI layer 3 tunnels and, with SDN, on layer 2 point-to-point connections. Most of the nations Internet ⑨ is moved into the cloud.

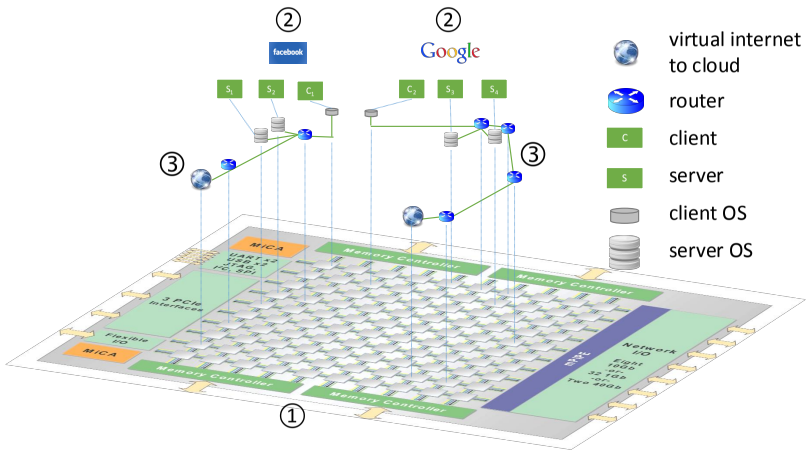


Figure 43

Multiple, in this case two, virtual internets extend from the cloud into the end-users equipment. The picture shows a multi-core CPU ① on which multiple service providers ② have instantiated virtual internets ③ connecting an on-CPU server farm that runs VMs containing services. This structure serves as a sandbox for the services. Furthermore, the service providers have instantiated on the CPU for each sandbox another VM containing client software to access these services. The client software also allows, if permitted, interactions with peripherals, such as a keyboard. In this end-system as well as in the cloud, virtual internets are isolated, making them very hard to attack, e.g., from the Internet.

BIBLIOGRAPHY

[1] R. J. Meijer, R. J. Strijkers, L. Gommans, and C. De Laat, "User programmable virtualized networks," in *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*. IEEE, 2006, pp. 43–43.

[2] R. Strijkers, M. X. Makkes, C. de Laat, and R. J. Meijer, "Internet factories: Creating application-specific networks on-demand," *Computer Networks*, vol. 68, pp. 187–198, 2014.

[3] R. Strijkers, L. Muller, M. Cristea, R. Belleman, C. De Laat, P. Sloot, and R. J. Meijer, "Interactive control over a programmable computer network using a multi-touch surface," *Computational Science–ICCS 2009*, pp. 719–728, 2009.

[4] B. Škorić and M. X. Makkes, "Flowchart description of security primitives for controlled physical unclonable functions," *International Journal of Information Security*, vol. 9, no. 5, pp. 327–335, 2010.

[5] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[6] T. Ignatenko, G. Schrijen, B. Skoric, P. Tuyls, and F. Willems, "Estimating the secrecy-rate of physical unclonable functions with the context-tree weighting method," in *Information Theory, 2006 IEEE International Symposium on*. IEEE, 2006, pp. 499–503.

[7] B. Škorić, "On the entropy of keys derived from laser speckle; statistical properties of gabor-transformed speckle," *Journal of Optics A: Pure and Applied Optics*, vol. 10, no. 5, p. 055304, 2008.

[8] P. Tuyls, B. Škorić, S. Stallinga, A. Akkermans, and W. Ophey, "Information-theoretic security analysis of physical uncloneable functions," in *Financial Cryptography and Data Security*. Springer, 2005, pp. 141–155.

[9] J. Buchanan, R. Cowburn, A. Jausovec, D. Petit, P. Seem, G. Xiong, D. Atkinson, K. Fenton, D. Allwood, and M. Bryan, "Forgery:'fingerprinting'documents and packaging," *Nature*, vol. 436, no. 7050, pp. 475–475, 2005.

[10] P. Tuyls, G. Schrijen, B. Škorić, J. Van Geloven, N. Verhaegh, and R. Wolters, "Read-proof hardware from protective coatings," in *Cryptographic Hardware and Embedded Systems-CHES 2006*. Springer, 2006, pp. 369–383.

[11] G. DeJean and D. Kirovski, "Radio frequency certificates of authenticity," Mar. 16 2010, uS Patent 7,677,438.

[12] B. Škorić, T. Bel, A. Blom, B. de Jong, H. Kretschman, and A. Nellissen, "Randomized resonators as uniquely identifiable anti-counterfeiting tags," in *Secure Component and System Identification Workshop, Berlin (March 2008)*, 2008.

[13] B. Gassend *et al.*, "Silicon physical unkown functions," in *Proc, 9th ACM Conf on Computer and Communications Security*, vol. 446, 2002.

[14] J. Guajardo, S. Kumar, G. Schrijen, and P. Tuyls, *FPGA intrinsic PUFs and their use for IP protection*. Springer, 2007.

[15] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Advances in cryptology-Eurocrypt 2004*. Springer, 2004, pp. 523–540.

[16] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM journal on computing*, vol. 38, no. 1, pp. 97–139, 2008.

[17] J. Linnartz and P. Tuyls, "New shielding functions to enhance privacy and prevent misuse of biometric templates," in *Audio- and Video-Based Biometric Person Authentication*. Springer, 2003, pp. 393–402.

[18] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Controlled physical random functions," in *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*. IEEE, 2002, pp. 149–160.

[19] P. Tuyls, B. Škoriç, and T. Kevenaar, *Security with noisy data: on private biometrics, secure key storage and anti-counterfeiting*. Springer Science & Business Media, 2007.

[20] B. Gassend, M. V. Dijk, D. Clarke, E. Torlak, S. Devadas, and P. Tuyls, "Controlled physical random functions and applications," *ACM Transactions on Information and System Security (TISSEC)*, vol. 10, no. 4, p. 3, 2008.

[21] E. Torlak, M. Van Dijk, B. Gassend, D. Jackson, and S. Devadas, "Knowledge flow analysis for security protocols," *arXiv preprint cs/0605109*, 2006.

[22] B. Gassend, M. van Dijk, D. Clarke, and S. Devadas, "Security with noisy data, chapter controlled physical random functions," 2007.

[23] M. X. Makkes, A.-M. Oprescu, R. Strijkers, C. de Laat, and R. J. Meijer, "Metro: Low latency network paths with routers-on-demand," in *Euro-Par 2013: Parallel Processing Workshops*. Springer, 2014, pp. 333–342.

[24] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[25] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding bgp misconfiguration," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 3–16, 2002.

[26] D. Johnson and D. Maltz, "Dynamic source routing in ad hoc wireless networks," *Kluwer International Series in Engineering and Computer Science*, pp. 153–179, 1996.

[27] X. Xiao, A. Hannan, B. Bailey, and L. Ni, "Traffic engineering with mpls in the internet," *Network, IEEE*, vol. 14, no. 2, pp. 28–33, 2000.

[28] Y. Yemini and S. Da Silva, "Towards programmable networks," in *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*. Citeseer, 1996, pp. 1–11.

[29] A. Nakao, L. Peterson, and A. Bavier, "Scalable routing overlay networks," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 49–61, 2006.

[30] D. N. B. Ta, T. Nguyen, S. Zhou, X. Tang, W. Cai, and R. Ayani, "Interactivity-constrained server provisioning in large-scale distributed virtual environments," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 2, pp. 304–312, 2012.

[31] T. A. Funkhouser, "Ring: a client-server system for multi-user virtual environments," in *Proceedings of the 1995 symposium on Interactive 3D graphics*. ACM, 1995.

[32] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker *et al.*, "Detour: Informed internet routing and transport," *Micro, IEEE*, vol. 19, no. 1, pp. 50–59, 1999.

[33] C. Ly, C. Hsu, and M. Hefeeda, "Improving online gaming quality using detour paths," in *Proceedings of the international conference on Multimedia*. ACM, 2010.

[34] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The tcp/udp bandwidth measurement tool," 2005.

[35] W. Barth, *Nagios: System and network monitoring*. No Starch Pr, 2008.

[36] "The NLNOG RING," http://ring.nlnog.net/.

[37] Z. Duan, Z. Zhang, and Y. Hou, "Service overlay networks: Slas, qos, and bandwidth provisioning," *IEEE/ACM Transactions on Networking (TON)*, 2003.

[38] N. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *Communications Magazine, IEEE*, vol. 47, no. 7, pp. 20–26, 2009.

[39] "Amazon EC2," http://www.amazon.com/ec2/.

[40] "Brightbox," http://www.brightbox.com.

[41] M. X. Makkes, A. Varbanescu, C. de Laat, and R. J. Meijer, "Fast packet forwarding engine based on software circuits," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*. ACM, 2015, p. 28.

[42] H. Bos, W. De Bruijn, M. Cristea, T. Nguyen, and G. Portokalidis, "Ffpf: Fairly fast packet filters," in *Proceedings of OSDI*, vol. 4, 2004, pp. 347–363.

[43] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Towards high performance virtual routers on commodity hardware," in *Proceedings of the 2008 ACM CoNEXT Conference*. ACM, 2008, p. 20.

[44] R. Bolla and R. Bruschi, "Pc-based software routers: high performance and application service support," in *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*. ACM, 2008, pp. 27–32.

[45] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "Routebricks: exploiting parallelism to scale software routers," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 15–28.

[46] S. Han, K. Jang, K. Park, and S. Moon, "Packetshader: a gpu-accelerated software router," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 195–206, 2010.

[47] A. Branover, D. Foley, and M. Steinman, "Amd fusion apu: Llano," *Micro, IEEE*, vol. 32, no. 2, pp. 28–37, 2012.

[48] M. Yuffe, E. Knoll, M. Mehalel, J. Shor, and T. Kurts, "A fully integrated multi-cpu, gpu and memory controller 32nm processor," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*. IEEE, 2011, pp. 264–266.

[49] K.-T. Cheng and Y.-C. Wang, "Using mobile gpu for general-purpose computing–a case study of face recognition on smartphones," in *VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on*. IEEE, 2011, pp. 1–4.

[50] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, *Scalable high speed IP routing lookups*. ACM, 1997, vol. 27, no. 4.

[51] D. E. White, *Bit-Slice Design Controllers and Alu's*. Garland Publishing, Inc., 1984.

[52] D. J. Bernstein and P. Schwabe, "New aes software speed records," in *Progress in Cryptology-INDOCRYPT 2008*. Springer, 2008, pp. 322–336.

[53] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 1998, pp. 1240–1247.

[54] J. Zhao, X. Zhang, X. Wang, Y. Deng, and X. Fu, "Exploiting graphics processors for high-performance ip lookup in software routers," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 301–305.

[55] Y. Li, D. Zhang, A. X. Liu, and J. Zheng, "Gamt: a fast and scalable ip lookup engine for gpu-based software routers," in *Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems*. IEEE Press, 2013, pp. 1–12.

[56] W. J. Dally, F. Labonte, A. Das, P. Hanrahan, J.-H. Ahn, J. Gummaraju, M. Erez, N. Jayasena, I. Buck, T. J. Knight *et al.*, "Merrimac: Supercomputing with streams," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. ACM, 2003, p. 35.

[57] M. V. Wilkes, "Edsac 2," *IEEE Annals of the History of Computing*, vol. 14, no. 4, pp. 49–56, 1992.

[58] M. Kwan, "Reducing the gate count of bitslice des." *IACR Cryptology ePrint Archive*, vol. 2000, p. 51, 2000.

[59] A. Munshi *et al.*, "The opencl specification," *Khronos OpenCL Working Group*, vol. 1, pp. l1–15, 2009.

[60] E. J. McCluskey, "Minimization of boolean functions*," *Bell system technical Journal*, vol. 35, no. 6, pp. 1417–1444, 1956.

[61] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff, "A medium-scale distributed system for computer science research: Infrastructure for the long term," *Computer*, vol. 49, no. 5, pp. 54–63, 2016.

[62] J. Shen, J. Fang, H. Sips, and A. L. Varbanescu, "An application-centric evaluation of opencl on multi-core cpus," *Parallel Computing*, vol. 39, no. 12, pp. 834–850, 2013.

[63] J. H. Lee, K. Patel, N. Nigania, H. Kim, and H. Kim, "Opencl performance evaluation on modern multi core cpus," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 1177–1185.

[64] A. Ali, U. Dastgeer, and C. Kessler, "OpenCL for programming shared memory multicore CPUs," in *5th Workshop on MULTIPROG, with HiPEAC (2012)*, January 2012.

[65] H. Song, F. Hao, M. Kodialam, and T. Lakshman, "Ipv6 lookups using distributed and load balanced bloom filters for 100gbps core router line cards," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 2518–2526.

[66] K. Sklower, "A tree-based packet routing table for berkeley unix." in *USENIX Winter*, vol. 1991, 1991, pp. 93–99.

[67] S. Nilsson and G. Karlsson, "Ip-address lookup using lc-tries," *Selected Areas in Communications, IEEE Journal on*, vol. 17, no. 6, pp. 1083–1092, 1999.

[68] V. Srinivasan and G. Varghese, "Faster ip lookups using controlled prefix expansion," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 1. ACM, 1998, pp. 1–10.

[69] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, *Small forwarding tables for fast routing lookups*. ACM, 1997, vol. 27, no. 4.

[70] T.-c. Chiueh and P. Pradhan, "High-performance ip routing table lookup using cpu caching," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 1999, pp. 1421–1428.

[71] Y. Zhu, Y. Deng, and Y. Chen, "Hermes: an integrated cpu/gpu microarchitecture for ip routing," in *Proceedings of the 48th Design Automation Conference*. ACM, 2011, pp. 1044–1049.

[72] M. X. Makkes, A. Taal, A. Osseyran, and P. Grosso, "A decision framework for placement of applications in clouds that minimizes their carbon footprint," *Journal of Cloud Computing*, vol. 2, no. 1, pp. 1–13, 2013.

[73] Matt Stansberry and Julian Kundritzki, "Uptime institute 2012 data center industry survey," http://uptimeinstitute.com.

[74] http://www.thegreenitreview.com/2013/05/european-data-centres-are-less-energy.html.

[75] D. J. Brown and C. Reams, "Toward energy-efficient computing," *Communications of the ACM*, vol. 53, no. 3, pp. 50–58, 2010.

[76] A.Taal, P. Grosso, and F. Bomhof, "Transporting bits or transporting energy: Does it matter?" http://www.surf.nl/nl/publicaties/Pages/TransportingBitsorTransportingEnergy.aspx.

[77] A. Taal, D. Drupsteen, M. Makkes, and P. Grosso, "Storage to energy: modeling the carbon emission of storage task offloading between data centers," in *Consumer Communications and Networking Conference (CCNC)*, 2014.

[78] IEA, "Co2 emissions from fuel combustion–highlights," *Paris http://www. iea. org/co2highlights/co2highlights. pdf. Cited July*, 2011.

[79] [online] http://en.wikipedia.org/wiki/Emission_intensity.

[80] Benjamin K. Sovacool, "Valuing the greenhouse gas emissions from nuclear power: A critical survey," *Energy Policy*, vol. 36, no. 8, pp. 2950–2963, 2008.

[81] http://ec.europa.eu/energy/energy_policy/doc/factsheets/mix/mix_nl_en.pdf.

[82] http://ec.europa.eu/energy/energy_policy/doc/factsheets/mix/mix_de_en.pdf.

[83] http://ec.europa.eu/energy/energy_policy/doc/factsheets/mix/mix_at_en.pdf.

[84] O. Rogers and D. Cliff, "Options, forwards and provision-point contracts in improving cloud infrastructure utilisation," *Journal of Cloud Computing*, vol. 1, no. 1, pp. 1–22, 2012.

[85] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 123–134, 2009.

[86] L. Rao, X. Liu, L. Xie, and W. Liu, "Minimizing electricity cost: optimization of distributed internet data centers in a multi-electricity-market environment," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[87] K. Kant, "Data center evolution: A tutorial on state of the art, issues, and challenges," *Computer Networks*, vol. 53, no. 17, pp. 2939–2965, 2009.

[88] J. Baliga, R. W.A. Ayre, K. Hinton, and R. S. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2011.

[89] R. S. Tucker, "Optical packet-switched wdm networks–a cost and energy perspective," in *Optical Fiber Communication Conference*. Optical Society of America, 2008.

[90] J. Baliga, R. Ayre, K. Hinton, and R. Tucker, "Energy consumption in wired and wireless access networks," *Communications Magazine, IEEE*, vol. 49, no. 6, pp. 70–77, 2011.

[91] http://www.surfnet.nl/en/Hybride_netwerk/SURFinternet/Pages/kaart.aspx#topologie.

[92] M. X. Makkes, R. Cushing, A. Belloum, S. Olabarriaga, M. Baranowski, C. de Laat, and R. J. Meijer, "Software defined internets," *IEEE Cloud Computing (Submitted)*, 2017.

[93] P. Hofer, F. Hörschläger, and H. Mössenböck, "Sampling-based steal time accounting under hardware virtualization," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ACM, 2015, pp. 87–90.

[94] A. G. Shoro and T. R. Soomro, "Big data analysis: Apache spark perspective," *Global Journal of Computer Science and Technology*, vol. 15, no. 1, 2015.

[95] W. Yang, X. Liu, L. Zhang, and L. T. Yang, "Big data real-time processing based on storm," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2013, pp. 1784–1787.

[96] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[97] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[98] R. Cushing, A. Belloum, M. Bubak, and C. de Laat, "Automata-based dynamic data processing for clouds," in *Euro-Par 2014: Parallel Processing Workshops*. Springer, 2014, pp. 93–104.

[99] "https://github.com/rstrijkers/sarastro [last accessed January 16, 2018]."

[100] "https://github.com/recap/pumpkin [last accessed January 16, 2018]."

[101] "https://github.com/marcccx/sdi [last accessed January 16, 2018]."

[102] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[103] A. Taal, M. X. Makkes, and P. Grosso, "Storage to energy calculator," in *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*. IEEE, 2014, pp. 1150–1151.

ACRONYMS

AC	Application Component
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
AS	Autonomous system
CPU	Central Processing Unit
DAS	Distributed ASCI Supercomputer
DNS	Domain Name Service
DWDM	Dense Wavelength Division Multiplexing
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
ICT	Information and Communication Technology
IP	Internet Protocol
ISP	Internet Service Provider
IaaS	Infrastructure as a Services
LAN	Local Area Network
MPLS	Multi Protocol Label Switching
NFV	Network Function Virtualization
NIC	Network Interface Card
NOP	No OPeration
OS	Operating System
OpenCL	Open Compute Language
OpenMP	Open Multi-Processing
PC	Personal Computer
PUE	Power Usage Effectiveness
PUF	Physical Unclonable Function
QoS	Quality of Service
RTT	Round Trip Time
SDI	Software Defined Infrastructure
SDK	Software Development Kit
TCP	Transmission Control Protocol

SDN Software Defined Networking
SIMD Single Instruction Multiple Data
SIMT Single Instruction Multiple Threads
SNE Systems and Network Engineering group of UvA
UPVN User Virtual Private Networking
UvA University of Amsterdam
VLAN Virtual Local Area Network
VM Virtual Machine
VOIP Voice Over IP
VPN Virtual Private Network

LIST OF PUBLICATIONS

JOURNAL PUBLICATIONS

1 B. Skoric and **M. X. Makkes**. Flowchart description of security primitives for Controlled Physical Unclonable Functions. International Journal of Information Security Volume 9, Number 5, P327-335.

2 **M.X. Makkes**, A. Taal, A. Osseyran, and P. Grosso. A decision framework for placement of applications in clouds that minimizes their carbon footprint. Journal of Cloud Computing, 2013. Springer

3 R. Strijkers, **M. X. Makkes**, C. de Laat, R.J. Meijer. Internet Factories: Creating Application-Specific Networks On-Demand. Elseviers journal of Computer Networks p187–198, 2014.

4 **M.X. Makkes**, R. Cushing, A. Belloum, S. Olabarriaga, M. Baranowski, C. de Laat, R.J. Meijer. Data Intrinsic Networked Computing. Submitted to IEEE Cloud Computing (Special Issue: Middleware for Multicloud).

5 A. Taal, **Marc X. Makkes**, M. Kaat, and P. Grosso.A multiple attribute relative quality measure based on the harmonic and arithmetic mean. Springerss journal of operational research.

6 S. Kovalchuk, D. Nasonov, A. Chirkin, M. Melnik, A. Visheratin, A. Belloum, **M.X. Makkes** Execution Time Estimation for Workflow Scheduling, the Journal of Future Generation of Computer Systems. Elsevier 2017.

CONFERENCE PAPERS

7 M.A. Kahlid, M. Marx, and **M.X. Makkes**. Entity models for trigger-reaction documents. The 8th Dutch-Belgian Information Retrieval workshop, 2008.

8 T. P. Jakobsen, **M. X. Makkes** and J. Dam Nielsen. Efficient implementation of the orlandi protocol. Proceedings of the 8th international conference on Applied cryptography and network security, Beijing, China. Lecture Notes in Computer Science vol 6123/2010, P255-272.

9 Z. Gong and **M. X. Makkes**. Hardware Trojan side-channels based on physical unclonable functions. Proceedings of the 5th IFIP WG 11.2 international conference on Information security theory and practice: security and privacy of mobile devices in wireless communication. Lecture Notes in Computer Science, 2011, Volume 6633/2011, P294-303.

10 Y. Demchenko, C. Ngo, **M. X. Makkes**, R. Strijkers, C. de Laat. Defining Inter-Cloud Architecture for Interoperability and Integration. The Third International Conference on Cloud Computing, GRIDs, and Virtualization, 2012 Nice France.

11 Y. Demchenko, **M.X. Makkes**, R. Strijkers, C. de Laat. Intercloud Architecture for Interoperability and Integration. IEEE 4th International Conference on Cloud Computing Technology and Science. 2012 Taipei, Taiwan

12 **M.X. Makkes**, C. Ngo, Y. Demchenko, R. Strijkers, R. Meijer, C. de Laat. Defining Intercloud Federation Framework for Multi-provider Cloud Services Integration. The Forth International Conference on Cloud Computing, GRIDs, and Virtualization, 2013 Valencia Spain. (Award: Best Paper)

13 C. Ngo, **M. X. Makkes**, Y. Demchenko, C. de Laat Multi-data-types Interval Decision Diagrams for XACML Evaluation Engine. Eleventh Annual International Conference on Privacy, Security and Trust (PST), 2013, Tarragona, Spain

14 **M.X. Makkes**, A. Oprescu, R. Strijkers and R.J. Meijer. MeTRO: Low Latency Network Paths with Routers-on-Demand. Euro-Par 2013: Parallel Processing Workshops - Large Scale Distributed Virtual Environments on Clouds and P2P. August 2013. Achen, Germany.

15 J. S. Van Der Veen, E. Lazovik, **M.X Makkes** and R.J. Meijer. Deployment Strategies for Distributed Applications on Cloud Computing Infrastructures. The 5th IEEE Conf. on Cloud Computing Technologies and Science (CloudCom2013), 2-5 December, Bristol, UK.

16 R. Strijkers, R. Cushing, **M.X. Makkes**, P. Meulenhoff, A. Belloum, C. de Laat, and R.J. Meijer. Towards an Operating System for Intercloud. The 3nd NetCloud2013 Workshop on Network Infrastructure Services as part of Cloud Computing, in Proc. The 5th IEEE Conf. on Cloud Computing Technologies and Science (CloudCom2013), 2-5 December, Bristol, UK.

17 **M. X. Makkes**, R. Cushing, A.M. Oprescu, R. Koning, P. Grosso, R.J. Meijer, and C. de Laat. Smart Cyber Infrastructure for Big

Data processing. . In Optical Fiber Communications Conference and Exhibition (OFC), 2014, pp. 1-3. IEEE, 2014.

18 A. Taal, D. Drupsteen, **M.X. Makkes**, P Grosso .Storage to Energy: modeling the carbon emission of storage task offloading between data centers. . IEEE 11th Consumer Communications and Networking Conference (CCNC), Las Vegas, US.

19 A. Taal, **M.X. Makkes**, and P. Grosso. Storage to energy calculator. Storage to energy calculator. IEEE 11th Consumer Communications and Networking Conference (CCNC) 2014, Las Vegas, US.

20 A. M Chirkin, A.S.Z. Belloum, S. V. Kovalchuk, **M.X. Makkes**. Execution time estimation for workflow scheduling. Workshop on Workflows in Support of Large-Scale Science, SuperComputing 2014, New Orleans, US.

21 **M. X. Makkes**, A. L. Varbanescu, C. de Laat, R.J. Meijer. Packet forwarding engine based on software circuits. ACM/IEE Computing Frontiers 2015, Ischia, Italy. (*Accepted for journal publication*

22 R.J. Meijer, R. Cushing, C. de Laat, P. Jackson, S. Klous, R. Koning, **M.X Makkes**, A. Meerwijk. Car2x with SOftware Defned Network, Network Fuctions Virtualization and Supercomputers: Technical and scientific preperation for the Amsterdam Areana Telecoms Fieldlab. 2014 International Conference on High Performance Computing & Simulation (HPCS), 2014 International Conference on

23 D. Frey, **M.X. Makkes**, P.L. Roman, F. Taïani, S. Voulgaris. Bringing secure Bitcoin transactions to your smartphone. The 15th Workshop on Adaptive and Reflective Middleware 2016.

24 **M.X. Makkes**, A. Uta, R. Bharath Das, V.N. Bozdog, H.E. Bal. P^2-SWAN: Real-time Privacy Preservering Computation for IoT Ecosystems. The International Conference on Fog and Edge Computing 2017, Madrid, Spain.

25 V. Bozdog, **M.X. Makkes**, A. Uta, R. Bharath Das, A. van Halteren. H.E. Bal. SwanLE: Exploiting Spatial Locality in Decentralized Sensing Environments. IEEE Ubiquitous Computing and Communications 2017, Guangzhou, China. (*Best paper award*)

26 R. Bharath Das, N. V. Bozdog, **Marc X. Makkes**, Henri Bal. Kea: A Computation Offloading System for Smartphone Sensor Data. 9th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2017), Hong Kong, China.

27 V. Bozdog, **M.X. Makkes**, H.E. Bal. RideMatcher: Peer-to-peer Matching of Passengers for Efficient Ridesharing. Submitted to CCGrid 2018.

OTHER PUBLICATIONS

- R. Cushing, **M.X. Makkes**, R. Strijkers and A. Belloum. Intercouds: Grid prothesis for workflow systems. The Fifth Ieee Internation Scalable Cloud Computing Challenge (SCALE 2012): http://www.cloudbus.org/ccgrid2012/cfp-scale.html

PROTOTYPES

P-1 Internet Factories: Configure/Dimension Cloud Resources for eScience Experiments. High Performance Computing and Simulation 2015, Amsterdam, NL

P-2 SuperComputing, SARnets 2014, New Orleans, USA

P-3 Building the Internet of the Future, COMMIT - The big future of data 2014, Amsterdam, NL

P-4 e-Factories 2013, SuperComputing, Denver, Co, USA

P-5 On-Demand Networks 2012, SuperComputing, SaltLake City, UT, USA

P-6 ICT.Delta 2012, Rotterdam, NL

P-7 Interactive Networks, LHC-ONE Workshop, Amsterdam, NL

P-8 Interactive Networks, SuperComputing 2011, Seattle, USA.

P-9 Interactive Network, International Next Generation Networks Workshop, Delft, 2011.

P-10 Energy calculator, the IEEE 11th Consumer Communications and Networking Conference.

PUBLICATION AUTHORSHIP

Author contributions to the publications used in this thesis.

Chapter 2 Flowchart description of security primitives for controlled physical unclonable functions [4]. Contributing other author: B. Škorić. B.Š. signaled the opportunity to improve upon the new cPUF concept from literature and invited M.X.M. to contribute cyber-security expertise, in particular that of VPNs. Then M.X.M. and B.Š. reverse engineered the complex pseudo code from the literature. M.X.M. and B.Š. iteratively extended that into flowchart descriptions of a system that combined cPUF security and VPN secure communications. The proof of security required M.X.M. and B.Š. to investigate the whole scope of attacker models. The initial version of the manuscript was created by M.X.M. and improved upon through various iterations between B.Š. and M.X.M. Published in International Journal of Information Security. B.Š. supervised the research.

Chapter 3 MeTRO: Low latency network paths with routers-on-demand [23]. Contributing other authors: A. Oprescu, R. Strijkers and R.J. Meijer. M.X.M. organized the support of and collaborated with members of the NLNOG Ring initiative. M.X.M. developed the path-finding algorithm, the network generating Netapp and designed, implemented, performed and analyzed the measurements. A.O. contributed to the manuscript. M.X.M. presented the work at Large Scale Distributed Virtual Environments on Clouds and P2P 2013. R.S. and R.J.M. supervised the research.

Chapter 4 Fast packet forwarding engine based on software circuits [41]. Contributing other authors: A. Varbanescu, C.T.A.M. de Laat, R.J. Meijer. M.X.M. invented and developed the new table lookup algorithm, implemented, performed and analyzed the experiments. A.V. aided in performing the experiments and contributed to the manuscript. M.X.M. presented the work at Computer Frontiers 2015. R.J.M. and C.d.L. supervised the research.

Chapter 5 A decision framework for placement of applications in clouds that minimizes their carbon footprint [72]. Contributing other authors: A. Taal, A. Osseyran and P. Grosso A. M.X.M. partially assisted in the iterative development, together with A.T. and P.G., of the carbon footprint models [77, 103]. M.X.M. demonstrated the energy calculator [103] and presented the energy models [77] at IEEE 11th Consumer Communications and Networking Conference (CCNC 2014). P.G. Supervised the research.

The work was published in Journal of Cloud Computing: Advances, Systems and Applications.

Chapter 6 Software controlled virtual infrastructure [92]. Contributing other authors: R. Cushing, A. Belloum, S. Olabarriaga, M. Baranowski, C. de Laat, and R.J. Meijer. M.X.M. implemented the control mechanisms and the Netapps, designed, performed the experiments and interpreted the results. R.C. implemented the Pumpking workflow manager. M.B. implemented the data filters. M.X.M. and R.C. prepared and implemented the SuperComputing demonstrations [**P-3, P-4**]. A.B. had an advisory role. C.d.L. and R.J.M. supervised the work. This work is submitted to IEEE Cloud Computing Magazin.

RESEARCH PROJECT ACKNOWLEDGEMENTS

This thesis research was supported the following organisations and projects:

- the Netherlands Organisation for Applied Scientific Research (TNO). TNO employed me in the period 2011-2015.
- the University of Amsterdam, in particular the System and Network Engineering (SNE) group of the Informatics Institute. This group, headed by prof. dr. ir. Cees T.A.M. de Laat, provided the inspiring scientific environment to do my research.
- the Dutch national program COMMIT/ in particular work package 2 of project P20, programmable infrastructures (PIF). This project, headed by prof. dr. ir. Henri E. Bal, provided the scientific context of the research.
- The Distributed ASCI Supercomputer (DAS) project. This project allowed me to access to their supercomputer to run the experiments conducted in Chapter 4.
- SURFnet, GIGAPort, and AgentschapNL. These organisations contributed to the research in Chapter 5.
- Secure Autonomous Response Networks (SARNET) project[1]. This project enabled me to draft the outlooks in Chapter 7.

1 https://sarnet.uvalight.net/

ACKNOWLEDGEMENTS

This thesis would not have been possible without the help and support of many people. First and foremost, I would like to thank my promotors Prof. dr. Robert J. Meijer and Prof dr. ir. Cees T.A.M. de Laat for giving me the opportunity to write this thesis, for their support and advice. Robert was my daily-supervisor with which I had many insightful discussions and joyful interactions about countless on and off-topic subjects. Cees was always ready and available to help either with research-related suggestions and comments on my work. Both provided many opportunities to present and demonstrate my research. Looking back on this period, I learned a lot from Rob and Cees.

Next, I would like to thank Rudolf Strijkers for bootstrapping my PhD en I enjoyed the intensive collaboration when we had to finish the demonstrations for SuperComputing. Rudolf and I had many interesting and fruitful discussions, during the long train journeys to Groningen and back. Very gratefully I am to all my co-authors for the numerous hours, days and weeks we collaborated on ideas, in random order: Paola, Ana L., Rudolf, Adam, Ralph, Reginald, Cahn, Yuri, Marijke, Boris, Anwar, Silvia, Mikolai, Jan-Sipke, Pieter, the Anas' and Arie.

Without my former employer, TNO, finishing my thesis would not have been possible and I would not have had such a wonderful time. TNO was the perfect place for me to explore and learn, especially because within TNO there is an expert on every topic. I would like to thank my two research managers Gert and Job who supported my research. In addition, I like to thank my colleagues at TNO, and especially Lenny for arranging the many requests that I had over the years.

For computational resources, I often made use of the DAS cluster. Without the help of Dennis Koelma and Kees Verstoep, running my experiments would not be so efficient. I am grateful that Henri Bal allowed me to finish my thesis while working at the Vrije Universiteit.

I wish to thank the people at the System and Network Engineering group at University of Amsterdam, in particular Arie, Paola, Adam, Jacqueline, Ralph, Paul and the Anas' for making this group pleasant work environment. I also want to thank the other former and current students for their company and interaction, Cosmin, Mathijs, Merijn, Rick and Reginald. For the many joyful discussions during lunchtime, drinks and meetings outside of university I would like than Kaveh, Alex, and Pieter.

I'm happy to have friends like Andrea, Arjan, Charlotte, Eric, Fieke, Glenn, Hilbert, Ilse, Klaartje, László, Lot, Marieke, Michiel, Monique,

Robin, Roel, William for their sympathies. Writing a PhD has lowlights and highlights.

I would not have been able to achieve all this without the unconditional love of my mother Lieselotte, my father Jaap, my sister Nicole, my grandma Aletta and my parents-in-law Ans, Evelyn, Hans and Jan and the rest of my family.

There are two little wonders, Julia and Luyt who bring so much joy, laughter and amazement to my life. They exhilarate my days and nights. Yet, I am most grateful to my partner Ilse, for her unlimited patience, support and love.

SUMMARY

This dissertation concludes that virtual internets are instrumental to the organization of globally distributed applications. In essence, a virtual internet comprises emulated internet subnets as well as emulated computer hardware. The emulated hardware runs the operating systems, software that implements the distributed application as well as network software, e.g., routers. The emulation of subnetworks is implemented via a set of IP-tunnels. Software called virtual machines (VM) emulate computer hardware. Currently, such emulations work very well and are deployed on large scale by companies for all kinds of purposes. Most software designed to use the Internet will operate with virtual internets too.

Virtual internets do have a geographical distribution. The distribution is determined by the geographical location of its VMs that can run on computers in data centers all over the world. With software one can manipulate the VMs and therefore also the software they execute. For instance, one can halt a VM and copy it entirely to another (hardware) computer on another location and continue processing. Such manipulations are used in this thesis to scale and distribute the applications.

Since Meijer et al. [1] the existence of virtual internets is known and since the SuperComputing demonstration **[P-5]** we knew how to generate them in large numbers and on large scale – we created [2] a 163-node virtual internet spanning the world. We also knew that the main application of virtual internet is to support distributed applications. Yet, at the beginning of the thesis research, essential functions for a practical deployment of virtual internets were missing. Concepts for security, optimal performance, distribution, CO_2 footprint, and scaling of virtual internets were not developed. To find them was the purpose of our research.

The inherent security of virtual internets is based on their isolation. They basically extent the insight of a VM being a sandbox to the insight that a set of VMs connected by a virtual internet forms a distributed sandbox. Only those engineers who have instantiated the virtual internet, know where it is, how to access it and can create the ability to transfer data from virtual internet to the internet and vice versa. To facilitate the most security-sensitive computation and communication processes on systems where all the privileged software (kernel, hypervisor, etc.) is potentially malicious, we developed the concept of controlled Physical Unclonable Functions (cPUFs). The concept describes a set of security primitives based on the intrinsic, unclonable, properties of physical devices, a kind of a device finger-

print. cPUFs are used to guarantee the integrity and confidentiality of computing and networking. For example, to give processes, bypassing the kernel, exclusive access to encrypted parts of memory. Via cPUFs one can be sure if, on which CPU, and what software has been executed in the virtual internet and that IP-tunnels are established to the right computers.

The use the virtual internets should not inflict performance penalties as that would limit their applicability. We showed that virtual internets links have a performance better or at least equal as the default internet path. Our research demonstrated software that, by trial and error, placed virtual routers in several data centers, connected them by IP-tunnels and found the best performing routes. The performance was in about half of the cases comparable to that of the default internet connection. In the other cases, the IP-tunnel network outperformed it. We also designed a novel routing-table lookup algorithm for software routers. Essentially the algorithm maps the table on assembler code in a CPU or GPU. This omits much of the relatively slow memory access, and that discriminates or algorithm from the other table lookup algorithms. In the case studied the performance improved by a factor ~10. For GPU's this resulted in a performance gain of almost a factor ~100. As table lookups are part of many computer programs, the significance of our novel lookup method extends to other software.

The virtual internet concept allows to develop software that generates a large, in terms of computer and network resources, distributed application. An issue in the deployment of such applications is the cost of execution. We have focused on an ecological cost, namely CO_2 emission. Combining details found in literature, we created and published an extensive model to compute this cost. We have shown how one calculates, for a desired cost, the maximum number of VMs on which to scale and distribute an application.

We devised an algorithm that, for a given maximum number of VMs, scaled the applications in a workflow to achieve free flow state of the workflow. With the free flow optimization criterion, all workflow applications process data in the same amount of time. That means that input queues in any application of the workflow do not grow, the resources are used most efficiently. As a side effect of this algorithm, the optimum geographical distribution is also achieved. Because of the scaling actions of the optimizing algorithm, the topology is sensitive to the relative performance of the instantiated workflow applications. Therefore, the topology of the workflow reflects properties of input data and processing speed of applications. Topology changes can be used to detect changes in these properties.

With virtual internets one can create a sandbox environment that minimizes the risk of abuse of shared data. Virtual internets would allow, for instance, to monitor and control parts of a complex ma-

chine by their manufactures whilst excluding all other operations on the data – e.g., to copy data to another location. Essential for this feature is that virtual internets can be used to enforce that data is shared only with a selected set of programs. Programs that can be certified to have no side effects. As the data exchange with the virtual internet is tightly controlled, data produced by any undesired activity of the programs remains confined to the virtual internet. One can deploy cPUF technologies to add pre-and post-conditions to the processing of data, e.g., a digital proof the execution of a program and check if the execution takes place on the specified CPU. Other software described in this thesis can scale and distribute this sandbox environment. Hence this thesis contributed significantly to a crucial Internet of Things issue: "How to share data safely"!

One can imagine to set up an IP tunnel from a laptop to each virtual internet that contains the services a person uses. For instance, a virtual internet for email, navigation, and for each banking and social media application. Then a glimpse for the future organization of the Internet emerges: countless virtual internets, probably called virtual subnetworks by network engineers, in cloud data centers. They are interconnected with end-systems, PCs, and other virtual internets via IP-tunnels. Most of the IP-tunnels reside in the cloud, some are connected via routers to the Internet that we know off now. Such a fine-grained subnet organization makes the scope of a security incident rather small, and is better tracked to its source. Hence we dare to forecast: "in future most of the Internet will reside in the cloud!"

SAMENVATTING

Dit proefschrift concludeert dat virtuele internetten instrumenteel zijn voor de organisatie van gedistribueerde applicaties. Een virtueel internet bestaat eigenlijk uit één of meer geëmuleerde internet subnetten en geëmuleerde computer hardware. Op de geëmuleerde hardware draaien besturingssystemen. Die besturingssystemen worden niet alleen gebruikt om de gedistribueerde applicatie zélf te executeren maar ook netwerksoftware, bijvoorbeeld routers. De emulatie van subnetwerken wordt geïmplementeerd met behulp van IP-tunnels. Computer hardware wordt geëmuleerd door software dat bekend is onder de noemer van virtuele machines (VM). Op dit moment is de kwaliteit van de emulatie hoog en worden VMs op grote schaal toegepast door bedrijven. Daardoor zijn bijna alle applicaties die voor het internet ontwikkeld zijn óók te gebruiken in virtuele internetten.

Virtuele internetten hebben een geografische spreiding dat bepaald wordt door de geografische ligging van de VMs. Die VMs kunnen draaien op computers in datacentra. Die datacentra bevinden zich over de hele wereld. Middels software kan men VMs manipuleren, en daarmee ook de software die zij executeren. Zo kan men een VM pauzeren, kopiëren en deze op een geheel andere (hardware) computer op een andere locatie weer verder executeren. Dergelijke manipulaties wordt in dit proefschrift gebruikt om gedistribueerde applicaties op te schalen en te distribueren.

Sinds Meijer et. al. [1] kennen we het bestaan van virtuele internetten en sinds de SuperComputing [**P-5**] demonstratie weten we hoe we deze in grote aantallen en op grote schaal kunnen genereren - honderden netwerkelementen verspreid over de hele wereld. We wisten ook dat in de implementatie van gedistribueerd applicaties de belangrijkste toepassing van virtuele internetten zou liggen. Maar essentiële functies voor zulke toepassingen ontbraken aan het begin van mijn onderzoek. Concepten voor beveiliging, optimale prestaties, distributie, kosten en het opschalen van virtuele internetten waren toen nog niet ontwikkeld. Het doel van mijn onderzoek was om deze concepten te vinden.

De inherente veiligheid van de virtuele internetten is gebaseerd op hun isolement. Alleen die ingenieurs die de virtuele internetten geconstrueerd hebben, weten waar ze zijn, hoe je data uitwisseling ermee regelt en hoe software de diensten van een virtueel internet kan aanspreken. Maar toch kan het zo zijn dat VM-software, besturingssystemen, softwarebibliotheken en communicatieprocessen zélf niet volledig veilig zijn. Daarom hebben wij het concept ontwikkeld van controlled Physical Unclonable Functies (cPUFs). Het concept baseert

securityprimitieven op intrinsieke, niet kopieerbare, eigenschappen van fysieke apparaten, dus op een soort vingerafdruk van de apparatuur. cPUFs worden gebruikt om de integriteit en vertrouwelijkheid van computers en netwerken te garanderen, bijvoorbeeld door processen, buiten het besturingssysteem om, exclusief toegang te geven tot versleutelde geheugengebieden. Verder kan men op basis van cPUFs bewijzen genereren dat software op een specifieke CPU uitgevoerd is en dat IP-tunnels met de juiste computers verbonden zijn.

De performance van virtuele internetten moet adequaat voor het gros van de denkbare toepassingen zijn. In ons onderzoek is software geconstrueerd dat een optimaal presterend virtuele internet pad kan maken. Die performance was gelijk en in de helft van de gevallen beter dan de normale Internetverbinding. Om dat voor elkaar te krijgen plaatste de software in diverse datacentra virtuele routers, verbond deze onderling met IP-tunnels en selecteerde de best presterende route. Ook hebben we een nieuwe methode voor het zoeken in routeertabellen uitgevonden. De traditionele methode doorzoekt de tabel die in het computergeheugen is geplaatst. De nieuwe methode beeldt de tabel af op assembler code in CPUs en GPUs. Wij vergeleken de prestaties van de nieuwe methode met die van de traditionele. De nieuwe methode leverde een versnelling van een factor ~10 voor CPUs en een factor ~100 voor GPUs. Deze tabel-doorzoek-methode is van groot belang, het doorzoeken tabellen vindt in vrijwel alle software plaats.

Het opschalen van een virtueel internet wordt door software gestuurd. Het resultaat kan een grote, in termen van benutte hoeveelheden fysieke computer- en netwerkapparatuur, gedistribueerde applicatie zijn, met navenante kosten. Wij hebben een inschatting gemaakt van CO_2 emissie, een ecologische kostenfactor. Deze speelt tegenwoordig een rol in de bepaling van de omvang en distributie van een applicatie. Bij grootschalige applicaties maakt het namelijk wél uit hoe de energiemix van diverse datacenters zijn, hoe deze en met welke netwerkapparatuur en bekabeling zij onderling verbonden zijn, hoe de efficiency van het gebruik van computers is, etc. We hebben daarom, op basis van een literatuurstudie, een model gepubliceerd dat de CO_2 kosten van virtuele internetten berekent. Dat stelt ons in staat om te bepalen hoeveel virtuele machines, en waar, we tegen welke kosten gebruiken gaan.

Vervolgens is een algoritme gemaakt en beproefd dat ter beschikking staande VMs optimaal gebruikt. Het algoritme streeft naar free flow tussen de onderdelen van de gedistribueerde applicatie. In de free flow toestand verwerken alle applicatieonderdelen de aangeboden data even snel. Mocht een onderdeel van de applicatie te langzaam werken, b.v. omdat de aard of hoeveelheid van de aangeboden data verandert, zorgt het algoritme voor een extra VM en een juiste aansluiting in de topologie van het netwerk in het virtuele internet.

Deze VM is óf ongebruikt óf wordt van een ander onderdeel van de applicatie weggenomen. Daarom weerspiegelt de topologie van de gedistribueerde applicatie, eigenschappen van de invoergegevens en verwerkingssnelheid van applicatie onderdelen. De topologische verandering kunnen gebruikt worden om verandering in deze eigenschappen waar te nemen. Een plezierig neveneffect van het free-flow algoritme is dat de optimale geografische spreiding ook wordt bereikt.

Met virtuele internetten kan men een omgeving creëren dat het risico van misbruik van gedeelde data minimaliseert. Met behulp van virtuele internetten kan een leverancier zijn machines besturen in een fabriek. Van belang voor de fabriek is dat hiermee geen bedrijfsgeheimen weglekken. In dit geval zal dan een virtueel internet alleen geauthentiseerde applicaties verbinden. Een virtueel internet isoleert namelijk dataverkeer van het Internet. De uitwisseling van gegevens door middel van virtuele internetten kan uitstekend worden gecontroleerd. Data dat toch door ongewenste software-activiteit geproduceerd wordt, blijft opgesloten in het virtuele internet. Met cPUF technologie kan men het verwerken van data van pre-en postcondities voorzien: bv een digitaal bewijs dat een programma is uitgevoerd op een specifieke CPU. Andere software, beschreven in dit proefschrift, kan deze data-deel-omgeving omgeving schalen en distribueren. Vandaar dat dit proefschrift aanzienlijk bijdraagt aan een essentieel Internet of Things issue: "Hoe kunnen we gegevens veilig delen"!

Men zou zich kunnen voorstellen dat elke dienst die iemand gebruikt, zoals email, navigatie, online banking en social media, appart kan in een virtueel internet beschikbaar wordt gesteld en appart met een IP-tunnel verbonden wordt met zijn laptop. Men ziet dan een glimp van de toekomstige organisatie van het Internet: talloze virtuele internetten in cloud datacenters, die waarschijnlijk door de netwerkingenieur virtuele subnetten worden genoemd. Zij worden verbonden door IP-tunnels met allerlei gecomputeriseerde apparaten, PCs en andere virtuele internetten. Sommige IP-tunnels staan via routers in verbinding met het Internet dat we nu kennen. De fijnmazige subnet structuur zorgt dat beveiligingsincidenten veel kleinere gevolgen hebben dan tegenwoordig. De bron daarvan kan om dezelfde reden beter worden getraceerd. Vandaar dat we de volgende voorspelling durven te maken: “in de toekomst zal het grootste deel van het internet zich in de cloud bevinden!”